Centre de recherche
informatique de Montréal

92-05131/

CRIM-90/12-02

Ada, un langage de développement

État de la question et recommandations

- Document de synthèse

Ettore Merlo, Ph.D.
Chercheur

Collection scientifique et technique
décembre 1990

Ont participé à la réalisation de ce rapport :

Dactylographie : **Henriette Bouchard**, secrétaire à la direction scientifique

Correction : **Susan Dubé**, responsable des communications
Jacqueline Bourdeau, directrice scientifique adjointe
Didier Clerc, directeur des laboratoires

Pour tout renseignement, n'hésitez pas à contacter :

Centre de recherche informatique de Montréal
Centre de documentation
3744, rue Jean-Brillant, bureau 500
Montréal (Québec)
H3T 1P1
Tél. : 340-5716
Fax : 340-5777

ISBN 2-921316-02-1
Dépôt légal - 4e trimestre 1990
Bibliothèque nationale du Québec
Bibliothèque nationale du Canada
©Centre de recherche informatique de Montréal

1. INTRODUCTION GÉNÉRALE 5
2. INTRODUCTION AU LANGAGE ADA 7
3. ASPECTS TECHNIQUES 9
3.1 Validation du compilateur 9
3.2 Procédure de validation 10
3.3 Rapport de validation VSR (Validation Summary Report) 11
3.4 Dépendances de l'implantation 12
3.5 Ada program design language 12
3.6 Environnements de développement et d'exécution 12
3.6.1 APSE 13
3.6.2 Système d'exécution 13
3.7 Ada et la portabilité 14
3.8 Ada et la réutilisation du logiciel 15
3.8.1 Comment Ada "supporte" la réutilisation 16
3.9 Considérations sur la réutilisation 16
3.10 Estimation de coûts 17
3.11 Gestion de configuration 18
3.12 Ada pour des systèmes en temps réel 18
3.13 Ada pour des systèmes distribués 20
3.14 Apprentissage d'Ada 20
3.14.1 Implications de formation 20
3.14.2 Indication pour l'évaluation de programmes 21
4. ÉVALUATION DU COMPILATEUR ET SÉLECTION 21
4.1 Besoins 21
4.1.1 Besoins en matériel pour l'évaluation 21
4.1.2 Besoins en logiciel pour l'évaluation 22
4.1.3 Besoins de l'ensemble de tests 23
4.2 Échéancier, cartes de temps, dépendances et coûts 23
4.3 Évaluation des vendeurs 24
4.4 Procédures d'évaluation 25
4.5 La technologie du compilateur 26
4.5.1 Évaluation d'options du compilateur et caractéristiques spéciales 28
4.5.2 Compilation incrémentielle 29
4.5.3 Efficacité en temps du code généré 29
4.5.4 Efficacité d'espace du code généré 30
4.6 Critères quantitatifs et évaluation des performances 30
4.7 Informations complémentaires 30
4.8 Réévaluation 31
4.9 Évaluation d'adaptation 31
5. ÉVALUATION DE PERFORMANCE 32
5.1 Performance de compilation et d'édition de liens 32
6. ÉVALUATION DE L'INTERACTION PERSONNE-MACHINE 34
6.1 Sorties d'information et diagnostics 34
6.2 Documentation 35
7. ÉVALUATION DES FACTEURS ENVIRONNEMENTAUX 36
7.1 Technologie de l'environnement 36
7.1.1 Outils du APSE 36
7.1.2 Bibliothèque de système 36
7.1.3 Support à l'éditeur de liens et au chargeur 37
7.1.4 Chargement sélectif 37
7.1.5 Le modèle du système d'exécution 37
7.1.6 Caractéristiques du système d'exécution 38

7.1.6.1 Efficacité de temps 39
7.1.6.2 Efficacité d'espace 39
7.1.7 Simulateur d'ordinateur d'exécution 40
7.1.8 Déchargement pour système d'exécution 40
7.1.9 Support pour la mise au point 41
7.1.10 Effet d'optimisation sur la mise au point 41
7.1.11 Mise au point sur l'ordinateur d'exécution 41
7.1.12 Dépendances de l'implantation 42
7.1.13 Traitement des interruptions 42
7.1.14 L'horloge et les aspects de synchronisation 42
8. ASPECTS D'ÉVALUATION DE PERFORMANCE 43
8.1 Stratégie pour l'évaluation des performances 43
8.2 Information sur les configurations standards d'évaluations de performance 44
8.3 Types de tests 44
8.4 Tests de caractéristiques de langage 45
8.5 Tests de capacité et de dégradation 45
8.6 Programmes de référence composites 45
8.7 Tests de référence synthétique 46
8.8 Tests spécifiques aux applications 47
8.9 Facteurs 47
8.10 Anomalies dans la synchronisation 48
8.11 Vérification de la synchronisation 48
8.12 Analyse des données et rapport d'évaluation 49
9. ENSEMBLE DE TESTS ET AUTRES TECHNOLOGIES DISPONIBLES 50
9.1 Le Ada Compiler Evaluation Capability (ACEC) 50
9.2 Les programmes de références PIWG 50
9.3 Le système d'évaluation Ada 50
9.4 Évaluation des références en Ada de l'Université du Michigan 50
9.5 Références de l'Aerospace Corporation 50
9.6 Dépôt logiciel Ada 51
9.7 Autres sources 51
10. COMPARAISON ENTRE ADA ET LES AUTRES LANGAGES 51
10.1 Interactions d'Ada avec d'autres langages 51
10.2 Interfaces du code Ada avec d'autres langages 52
10.3 Remplacement de la totalité du système 52
10.4 Traduction automatique 53
10.5 Langage de traduction 53
11. ORIENTATIONS PRATIQUES 54
11.1 Scénario d'utilisation 54
11.2 Stratégie de départ 55
11.3 Compilateurs 56
11.4 Processus de sélection 56
11.5 Réutilisation 57
11.6 Systèmes en temps réel 57
BIBLIOGRAPHIE 59
ACRONYMES 63
ANNEXE 1 - RAPPORT DE COLLOQUES
ANNEXE 2 - THE SEI RESIDENT AFFILIATES PROGRAM
ANNEXE 3 - LISTE DE LA DOCUMENTATION DISPONIBLE SUR LES VENDEURS

1. INTRODUCTION GÉNÉRALE

Le présent document contient une discussion de caractère didactique sur les implications du choix de Ada comme langage de développement. Cette discussion touche les aspects généraux d'Ada, les critères de choix d'un environnement et d'un système de compilation, et les aspects d'évaluation d'un système de développement et de ses performances. Ce document a été rédigé de façon rigoureuse en cherchant à garder le langage fluide et autant que possible informel. La motivation était de fournir aux intéressés un document à large spectre contenant des indications pour le développement de plans basés sur Ada. La discussion sur l'évaluation et la sélection du compilateur a été profondément inspirée par les publications [FORE87] et [WEID89] du SEI, qui devraient toujours être consultées pour plus de détails.

Le rapport s'adresse à des personnes qui veulent démarrer des activités Ada avec peu ou pas d'expérience préliminaire. Ces personnes peuvent être des gestionnaires de projet qui ont besoin de comparer les bénéfices et les risques que comporte l'utilisation d'Ada, des programmeurs, des analystes ou des chercheurs qui désirent s'introduire dans le monde Ada.

La section 2 introduit brièvement le langage Ada, la section 3 présente les aspects techniques de validation des compilateurs Ada, des documents de validation, de l'environnement Ada, de la réutilisation etc. La section 4 traite de l'évaluation du compilateur et des critères pour sa sélection. Les sections 5, 6, 7 et 8 présentent les aspects d'évaluation de performance, d'interaction personne-machine, et des facteurs environnementaux. La section 9 contient une liste commentée des programmes de référence et autres technologies disponibles. La section 10 effectue la comparaison entre l'utilisation d'Ada et d'autres langages de programmation, et la section 11 présente des orientations pour l'utilisation pratique d'Ada. Des références bibliographiques qui sont à considérer comme sources d'information plus détaillée sont contenues dans la bibliographie. L'annexe 1 contient le rapport sur les colloques ACM SIGAda-1989 et TRI-Ada-1989, l'annexe 2, le programme d'affiliation au SEI et l'annexe 3, la liste de la documentation disponible sur les vendeurs.

2. INTRODUCTION AU LANGAGE ADA

Les directives de la Défense américaine requièrent l'utilisation d'Ada pour des applications spécialisées et intégrées aux équipements, et pour les applications essentielles aux missions. Selon la directive DoD 3405.1 du ministère américain de la Défense, Ada sera le seul langage commun de programmation pour les ressources d'informatique de la défense qui sont utilisées dans des systèmes intelligents pour la commande et le contrôle de forces militaires, ou comme partie intégrale d'un système d'armes. Une autre directive (DoD 3405.2) requiert l'utilisation d'un compilateur Ada validé et des principes de génie logiciel facilités par Ada, ainsi que l'application d'Ada comme un langage de description de la conception PDL. Des exceptions seront acceptées pour ces directives, mais il sera de plus en plus difficile de les obtenir. Dans la première directive, on lit qu'Ada devra être utilisé pour toutes les applications sauf lorsque l'utilisation d'un autre langage approuvé d'ordre supérieur est plus rentable selon des critères d'efficacité des coûts du cycle de vie de l'application. Le but à long terme est d'établir Ada comme le langage premier d'ordre élevé du ministère de la Défense.

Plusieurs études provenant d'industries et du ministère américain de la Défense ont montré que le logiciel est en train de prendre une tranche de plus en plus grande dans le coût et la complexité d'un système de défense, et que la demande pour du logiciel de qualité a monté plus vite que la capacité de le produire. Le ministère américain de la Défense a établi les programmes Ada, STARS (Software Technology for Adaptable for Reliable System) et le "Software Engineering Institute" pour étudier le problème. Les conditions d'affiliation au SEI sont inscrites à l'annexe 2.

Ada a été aussi développé pour réduire le nombre de langages de programmation du ministère de la Défense et pour encourager l'utilisation des méthodes modernes de développement du logiciel. Les efforts concernant Ada sont gérés par le "Ada Joint Program Office" (AJPO).

Au 1er février 1989, la liste officielle des compilateurs validés, disponible à l'AJPO, comportait 164 compilateurs de base validés et 70 compilateurs

dérivés pour plus de 50 vendeurs et des dizaines de machines différentes. La liste complète et mise à jour régulièrement est disponible à travers les réseaux à partir du CRIM.

En conséquence, il ne devrait pas être nécessaire de construire un compilateur complètement nouveau pour le projet envisagé. Toutefois, deux actions pourraient devenir nécessaires: le "retargetting" (avec du temps et avec des coûts variant entre 750 000 $ à plus d'un million et des temps de développement de 9 à 15 mois) et les améliorations spécifiques au projet avec des coûts et des temps variables. En ce qui concerne l'impact sur le matériel, les compilateurs Ada demandent plus de ressources de calcul que ceux de la majorité des autres langages. Cependant, ils peuvent détecter beaucoup plus d'erreurs de programmation et fournissent un niveau de support beaucoup plus élevé que d'autres langages pour bâtir des grands systèmes. Malheureusement, même si l'exécution d'un programme Ada devrait être aussi efficace que des programmes dans d'autres langages, actuellement, il s'avère que les compilateurs orientés vers des machines avec des ressources limitées n'ont pas encore atteint ce niveau de maturité.

Le plus grand avantage d'Ada est qu'il fournit les moyens pour obtenir les bénéfices des méthodes modernes de génie logiciel: la portabilité du code, les ressources humaines, la maintenabilité, la fiabilité, une base commune pour outils et méthodologie, la gestion de la complexité, la modularité, la visibilité et la productivité pour les gestionnaires. Obtenir les bénéfices Ada veut dire investir dans la formation, le développement d'outils, dans des projets pilotes et dans le développement standard pour obtenir des bénéfices pendant tout le cycle de vie du système. L'importance d'Ada dans le support au génie logiciel a été reconnue même en dehors du ministère de la Défense. Des engagements visant l'utilisation d'Ada dans le développement commercial ont été pris par des compagnies comme Computer Corporation of America, Shell Oil, Boeing Airplane Company, Nippon Telephone & Telegraph et Philips. De même en Europe, 80% de l'utilisation d'Ada est de nature commerciale.

Dans l'avenir, on bénéficiera d'une réduction de coût provenant de l'utilisation d'un seul langage moderne d'ordre élevé. Évidemment, il faut

compter des coûts initiaux plus élevés pour l'acquisition des outils informatiques, pour la formation et pour le développement du matériel. Toutefois, à long terme, l'utilisation d'Ada est censée réduire les coûts d'entretien et de développement des projets singuliers, suite à l'utilisation de meilleures méthodes de développement.

3. ASPECTS TECHNIQUES

Les compilateurs Ada sont disponibles maintenant pour des applications exécutées sur des ordinateurs universels qui n'ont pas de limitations sérieuses de mémoire ou de performance. Chaque projet doit quand même évaluer la qualité et la fiabilité des compilateurs en observant l'efficacité de compilation et du code objet, les services additionnels du compilateur et les supports qui ont besoin de systèmes spécialisés intégrés aux équipements. En ce qui concerne les techniques d'évaluation, se référer aux chapitres 4 à 9.

3.1 Validation du compilateur

La validation des compilateurs Ada est un processus développé par le AJPO pour tester la conformité d'un compilateur Ada à la définition du langage, le standard ANSI/MIL-STD-1815A. Pour être validé, un compilateur Ada doit passer un test standard connu comme la capacité de validation de compilateurs Ada ACVC (Ada Compiler Validation Capability). Le processus de validation ne teste pas des aspects de performance comme la vitesse du compilateur ou l'efficacité du code généré. La compilation n'implique pas qu'un compilateur est adéquat du côté de la qualité et des caractéristiques pour l'utilisation dans un programme spécifique.

Les tests de validation devraient être exécutés de nouveau sur chaque compilateur qui a été modifié pour garantir qu'il n'y a pas de déviation par rapport au standard. La validation est un processus qui est fait pour tester la conformité d'une implantation Ada à la définition du langage spécifié dans le standard ANSI/MIL-STD-1815A. La performance en compilation et en exécution, la capacité, les options pour l'utilisateur et beaucoup d'autres aspects qui ne sont pas inclus dans le standard rendent le compilateur adéquat

ou non au projet. Certains compilateurs sont plus appropriés pour des buts éducationnels, d'autres le sont pour des applications de systèmes de gestion des informations et certains autres sont plus adaptés pour des applications en temps réel.

Le gouvernement du Canada fait référence aux organismes américains pour les procédures de validation des compilateurs. Les procédures et les indications officielles pour la validation sont contenues dans un document de 25 pages nommé Ada Compiler Validation Procedures and Guidelines paru le 1er janvier 1987. Le ACVC (Ada Compiler Validation Capability) est un système pour vérifier la conformité des compilateurs par rapport au standard. Le AVS (Ada Validation Suite), qui fait partie du ACVC, est un ensemble de programme de tests qui est mis à jour annuellement. La version ACVC 1.10 a été rendue opérationnelle le 1er juin 1988. La version 1.11, livrée pour commentaires publics et révisions le 1er juin 1989, est devenue opérationnelle le 1er décembre 1989 et expirera le 1er juin 1991. Il doit être compris bien clairement que la validation n'a jamais eu l'intention de mesurer l'adaptabilité pour l'utilisation. Le test ACVC ne vérifie pas systématiquement les aspects de capacité ou de performance. Malgré l'opinion commune, le standard oblige un compilateur à implanter toutes les clauses de représentation et les caractéristiques dépendant de l'implantation décrites dans le chapitre 13 du manuel de référence. La version 1.10 du ACVC inclut plusieurs tests des caractéristiques du chapitre 13. Un autre aspect subtil du test ACVC est que le vendeur d'un compilateur est censé tester le compilateur en utilisant seulement un ensemble d'options du compilateur, alors qu'en fait le compilateur peut fournir plusieurs ensembles d'options. Ces options et leur interaction mutuelle ont le potentiel d'introduire des erreurs dans le code généré. Le test ACVC cherche seulement à déterminer la conformité au standard du langage; et il ne touche pas d'autres aspects d'évaluation.

3.2 Procédure de validation

La procédure courante pour un vendeur de compilateur est d'obtenir l'ensemble de tests, d'indiquer ses intentions de valider son compilateur avec une des facilités de validation (AVF) d'Ada, aux États-Unis ou en Europe.

Un compilateur qui est validé selon la procédure est appelé un compilateur de base. Un compilateur dérivé peut être un des suivants: un compilateur de base sur une configuration équivalente (même ordinateur et système d'exploitation), un compilateur modifié sur une configuration de base ou un compilateur modifié sur une configuration équivalente. Pour des compilateurs qui sont dérivés des compilateurs de base, il existe une procédure d'enregistrement qui transfère le statut de validation sans en exécuter toutes les étapes. Le statut d'un compilateur dérivé comme compilateur validé peut être mis en cause; si l'accusation est fondée, le vendeur doit corriger les erreurs dans les 30 jours ou le compilateur perd son statut de validation.

3.3 Rapport de validation VSR (Validation Summary Report)

C'est un document très important décrivant le processus de validation pour les usagers en train d'évaluer des compilateurs Ada. Ces rapports incluent les composantes suivantes:

- Déclaration de conformité.
- Description de tous les tests ACVC qui ont été exécutés sur les compilateurs de base.
- Une table qui montre la classe et la catégorie de tous les tests ACVC et leur résultat.
- Description de l'environnement de test.

Le VSR reflète toutes les décisions qui ont été prises concernant les aspects des tests contestés. Le VSR inclut les options qui dépendent de l'implantation devant aussi être fournies à l'annexe F du manuel de référence du vendeur.

3.4 Dépendances de l'implantation

Il y a une grande abondance de variations qui dépendent de l'implantation permise dans le langage Ada standard. L'impact de performance sur les utilisateurs de compilateurs est important. Le ARTEWG "Ada Runtime Environment Working Group" est une source d'information sur les dépendances d'implantation. Le Uniformity Rapporteur Group (URG) en est un autre. Le ACEC (Ada Compiler Evaluation Capability) teste 24 catégories d'optimisation. Les clauses de représentations et des caractéristiques dépendantes de l'implantation sont contenues dans le chapitre 13 du manuel de référence Ada. Pour plusieurs applications, ces aspects ne sont pas pertinents; mais pour des systèmes spécialisés et intégrés à l'équipement, ils sont essentiels. Le test plus récent Ada ACVC Version 1.10 comporte une série de tests pour vérifier les besoins mentionnés au chapitre 13; toutefois ce n'est pas une couverture complète.

3.5 Ada program design language

Le Ada Program Design Language (PDL) est une notation formelle utilisée pour capturer des décisions de conception. Les caractéristiques Ada au niveau des programmes sont aussi utilisables dans la conception pour décrire l'architecture du logiciel, c'est-à-dire la structure du système, ses interfaces, son comportement et la documentation de la conception. Chaque langage de description de conception devrait être utilisé comme une partie de la méthodologie générale de développement.

3.6 Environnements de développement et d'exécution

Dans le développement Ada, deux types d'ordinateurs sont utilisés: l'ordinateur de développement et l'ordinateur d'exécution. Même si les deux activités peuvent être réalisées sur la même machine, l'ordinateur de développement est souvent différent de l'ordinateur d'exécution parce que plusieurs autres ressources sont requises pour développer et compiler du code. L'ordinateur d'exécution est fréquemment spécialisé et intégré à l'équipement et fonctionne comme une partie intégrale d'un système d'armes,

comme par exemples des ordinateurs pour des robots industriels, des systèmes de navigation et de l'équipement de contrôle de processus.

3.6.1 APSE

APSE est un terme utilisé pour Ada Programming Support Environment. Un système de compilation Ada n'inclut pas seulement un compilateur. Il y a un ensemble d'outils minimal sans lequel le compilateur devient presque inutilisable. Cet ensemble est appelé MAPSE (Minimal Ada Programming Support Environment). Parmi ces outils, on peut inclure la bibliothèque du système, l'éditeur de liens et le chargeur, le "debugger" et le simulateur d'ordinateur d'exécution. Les trois premiers sont normalement considérés comme faisant partie du MAPSE, le quatrième est considéré important seulement pour le système de développement et d'exécution différents. Ces outils sont parmi les plus importants et parmi ceux qui sont couplés le plus étroitement avec le compilateur même. Même si un APSE n'est pas nécessaire quand on utilise Ada, les outils du MAPSE nécessitent toutefois une considération particulière par le gestionnaire du projet. Le CAIS est un standard d'interfaces proposé pour la majeure partie des fonctions du système qui a été développé pour faciliter la transportabilité des outils entre des APSE. Il n'existe présentement que peu d'implantations-prototypes de CAIS.

3.6.2 Système d'exécution

Le logiciel d'exécution fournit les fonctions pour le support additionnel requis pour l'exécution des programmes Ada sur une architecture particulière. Les logiciels Ada d'exécution disponibles pour un système universel ne répondent pas nécessairement à tous les besoins spécifiques du logiciel pour la défense. Des modifications aux logiciels d'exécution pourraient être nécessaires. Le logiciel d'exécution Ada est normalement en dehors du contrôle du développeur du système, étant donné qu'il est réalisé et entretenu par le vendeur de compilateur Ada, et s'avère souvent sa propriété. Lors de modification du logiciel d'exécution, le gestionnaire du programme devrait être attentif aux points suivants: le budget nécessaire, l'impact des modifications sur la validation, les coûts d'entretien récurrent si et seulement

si les modifications sont faites par les développeurs après l'acquisition des droits nécessaires.

3.7 Ada et la portabilité

Ada est un langage qui a pour caractéristique importante de promouvoir la portabilité du logiciel. Cette portabilité décrit l'ampleur des modifications selon lesquelles le logiciel peut être porté d'un ordinateur à un autre. Un aspect de la portabilité est de savoir si le code de l'application peut être porté sur un autre système de développement ou d'exécution, ou si le code existant peut être porté sur un autre système de compilation pour le même ordinateur d'exécution. Un changement dans le système de compilation peut être requis pour un certain nombre de raisons incluant le besoin additionnel de performance, le besoin de fonctionnalités additionnelles, le manque de support ou le manque d'activité de développement de la part du vendeur. L'usager doit reconnaître que des compilateurs hautement adaptés aux besoins peuvent être la seule façon de satisfaire des contraintes de performance, mais que chaque caractéristique spéciale qui est incluse peut devenir un obstacle à la portabilité sur un nouveau système. Il est aussi important de comprendre si les besoins de performance peuvent être satisfaits dans un certain système de compilation sans utiliser des caractéristiques qui dépendent de l'implantation. Dans le cas contraire, l'usager doit être au courant des contraintes de portabilité: il doit isoler les dépendances de l'implantation dans de plus petits paquetages et les documenter soigneusement pour faciliter le transport.

Ada se compare favorablement à d'autres langages, en ce qui concerne la portabilité, parce que:

- Il fait respecter le standard à travers la validitation du compilateur.

- Les caractéristiques de langage comme les paquetages et les types privés permettent au développeur du logiciel d'isoler les dépendances de la machine du reste du logiciel. Les

modifications requises en cas de transport sont centralisées et la probabilité d'erreurs est plus faible.

Ada favorise donc la portabilité qui, par contre, n'est pas automatique. En particulier, elle peut être mise en danger par les compromis faits entre le maximum d'efficacité de performance et l'utilisation des caractéristiques du langage dépendantes de l'implantation.

Comme pour la portabilité, la standardisation du langage Ada et certaines caractéristiques du langage sont particulièrement adaptées pour promouvoir la réutilisation du logiciel. Toutefois Ada ne rend pas automatiquement le logiciel réutilisable; il possède certaines caractéristiques dans le langage Ada qui rendent la réutilisation possible. La réutilisation est intéressante parce que plusieurs fonctions dans un nouveau système sont semblables sinon identiques à celles développées préalablement. La réutilisation du code existant et de la conception existante, et ensuite de son application, ont le potentiel de réduire d'une façon significative les coûts de développement d'un programme et de produire des systèmes plus fiables.

3.8 Ada et la réutilisation du logiciel

Plusieurs fonctions dans un nouveau système sont semblables sinon identiques à celles développées préalablement dans d'autres systèmes. Si le logiciel était conçu de la façon appropriée, ces fonctions pourraient être réutilisées pour produire plus rapidement de nouveaux systèmes, plus fiables et moins coûteux. La réutilisation du logiciel n'est pas un nouveau concept mais jusqu'à maintenant, des obstacles l'ont empêchée de devenir une réalité. Parmi ceux-ci, notons:

- Absence de motivation.
- Manque d'évolution du processus de développement de logiciel. Le logiciel était considéré comme un objet impossible à gérer.

- Langages techniquement limités.
- Manque de standardisation des langages.
- Obstacles sociaux et culturels.
- Problèmes avec l'indexage et le repérage.
- Efficacité.

3.8.1 Comment Ada "supporte" la réutilisation

Ada est spécialement adapté pour promouvoir la réutilisation du logiciel, à cause de sa standardisation et de certaines de ses caractéristiques spéciales. Toutefois, l'utilisation d'Ada ne produit pas du logiciel automatiquement réutilisable. Il existe des mécanismes possibles pour concevoir une composante réutilisable. En fait, la programmation pour la réutilisation requiert un niveau plus élevé d'effort que la programmation standard.

3.9 Considérations sur la réutilisation

Avant que la réutilisation extensive puisse devenir une réalité, des considérations contractuelles et de gestion doivent être adressées parmi les autres:

- Droits sur les données. Qui est le propriétaire du logiciel original, spécialement celui qui a été développé sous contrat?
- Estimation de coût. Quel impact la réutilisation aurait sur les procédures d'estimation de coût?
- Qui corrige les erreurs dans du code réutilisé?

- Responsabilité du contrat. Si un contracteur a des problèmes pour faire fonctionner les codes réutilisés, qui en est responsable?

- Motivation. Qui paie pour les efforts incluant le test additionnel du logiciel réutilisable? Quel changement culturel est nécessaire pour développer les motivations à ce sujet?

3.10 Estimation de coûts

Des enregistrements prouvés de coûts, horaires, qualité et expertise dans les domaines d'application sont importants. Plusieurs modèles connus d'estimation des coûts comme Cocomo, Softcost, etc. sont actuellement en révision pour rendre compte de l'expérience avec Ada, sauf en ce qui concerne le début des activités. Les gestionnaires de projets peuvent aider à raffiner ultérieurement les modèles d'estimation des coûts en recueillant et en partageant des données sur ce sujet. Plusieurs études montrent que la productivité est généralement plus élevée avec Ada qu'avec d'autres langages, On rapporte, sous toute réserve, des améliorations de l'ordre d'un facteur 2. Il est normal que la productivité au début d'un projet soit basse en raison du temps d'apprentissage. Les chiffres de productivité peuvent varier selon les capacités des personnes, la taille et la grandeur du système en développement, le degré de consistance dans les spécifications des besoins, la maturité et la sophistication de l'environnement de développement, et la documentation sur les besoins.

Dans les modèles d'estimation des coûts, un paramètre très important est la taille du système, mesurée en nombre de lignes de code source. Il est difficile de comparer le langage Ada à d'autres langages parce que les opinions diffèrent sur la définition d'une ligne de code source et comment utiliser les valeurs calculées. En attendant que les modèles d'estimation des coûts soient revus pour Ada, on peut donner les conseils suivants:

- Utiliser une métrique consistante quand on calcule les lignes de code. Compter les points-virgules à la fin des instructions est une mesure Ada généralement acceptée.

- Considérer les déclarations Ada comme lignes de code.

- Recueillir des données, de façon à calibrer les nouveaux modèles de productivité de lignes de code. Des sources de renseignement Ada sont le AJPO et le Software Engineering Institute.

3.11 Gestion de configuration

La gestion de la configuration disciplinée est un préalable à la gestion effective du logiciel Ada. À petite échelle, la gestion de configuration est supportée par des outils traditionnels orientés vers les fichiers et elle est pratiquement indistincte d'un langage de programmation à un autre. Quand le système devient plus grand, la gestion de configuration est un problème qui devient beaucoup plus complexe indépendamment du langage. Deux actions sont importantes pour la gestion de configuration des grands systèmes Ada: la première est de vérifier que la librairie de programme Ada fournit un support pour la recompilation automatique; la seconde est d'évaluer le niveau d'intégration entre le compilateur Ada et l'outil de gestion de configuration sélectionné.

3.12 Ada pour des systèmes en temps réel

Des inquiétudes ont été soulevées sur l'utilité d'Ada dans la programmation des systèmes en temps réel. Ces inquiétudes portent sur les points suivants:

- Efficacité.

- Certaines implantations Ada ne traitent pas les interruptions avec une efficacité acceptable.

- Certaines implantations Ada prennent trop de temps pour changer d'exécution d'une tâche à une autre.

- Le code généré par certains compilateurs Ada est moins efficace en terme de temps et d'espace que celui généré par les compilateurs d'autres langages.

- Contrôle de synchronisation.

- Quand une activité a besoin d'être exécutée à une fréquence périodique, il semble être difficile de contrôler la fréquence avec une précision suffisante.

- Les tâches Ada sont exécutées sous le contrôle du logiciel d'exécution qui est écrit par le vendeur du compilateur. Si ce logiciel ne rencontre pas les besoins d'une application particulière en temps réel, il pourrait être difficile ou dispendieux de modifier le système d'exécution.

- Il peut être difficile d'analyser les comportements temporels d'un système de tâches Ada et ce comportement peut dépendre aussi d'une façon significative du logiciel d'exécution qui n'est pas sous le contrôle direct du programmeur de l'application.

Ces inquiétudes s'appliquent généralement aux implantations courantes des compilateurs, des systèmes d'exécution mais pas au langage. Ils ne veulent pas nécessairement dire qu'Ada ne peut pas être utilisé pour des systèmes en temps réel. Le fait qu'Ada possède des caractéristiques pour le traitement concurrent ne veut pas dire que ces fonctions doivent être utilisées pour les systèmes en temps réel. La majeure partie des bénéfices d'Ada peut être obtenue pour des systèmes en temps réel même si les tâches Ada ne sont pas utilisées. Si les tâches Ada ne sont pas utilisées, il est important de vérifier que:

- L'implantation du compilateur ne charge pas automatiquement le support d'exécution non nécessaire aux tâches afin de sauver de l'espace mémoire.

- Le besoin d'utiliser un système d'exécution en temps réel existant, ou d'écrire un système d'exécution pour des buts spéciaux soit inclus comme une partie du plan de chaque projet.

L'efficacité du code est censée s'améliorer considérablement au fur et à mesure que les compilateurs acquièrent de la maturité. Les problèmes reliés au contrôle de synchronisation sont plus difficiles à résoudre. La réponse à court terme est simplement d'éviter d'utiliser les tâches Ada pour des systèmes qui ont des besoins critiques en temps réel.

3.13 Ada pour des systèmes distribués

Une des façons d'utiliser Ada dans des systèmes distribués est d'écrire des programmes Ada indépendants qui sont exécutés sur chaque noeud du système. Le logiciel présent sur les noeuds différents communiquerait avec un protocole de traitement des messages implanté en dehors du langage Ada. Les avantages de l'utilisation d'Ada sur d'autres langages seront une facilité d'entretien améliorée et la réduction des coûts à long terme.

3.14 Apprentissage d'Ada

3.14.1 Implications de formation

Pour maximiser les bénéfices qu'on peut retirer d'Ada, il faut connaître les techniques de génie logiciel ainsi que le langage. Il est important d'enseigner le génie logiciel en même temps que le langage, en prévoyant que les coûts associés à la production en Ada seront plus élevés que pour d'autres langages.

3.14.2 Indication pour l'évaluation de programmes

Les aspects les plus importants de tout programme sont l'emphase sur le génie logiciel et la pratique. L'emphase des cours devrait porter sur la compréhension de ce que le langage peut faire et comment il devrait être utilisé pour produire des systèmes bien conçus en termes d'ingénierie. Les cours qui enseignent seulement la syntaxe et la sémantique, c'est-à-dire la grammaire d'Ada, devraient être évités. Il est très important que des exercices de programmation pratique soient considérés comme une composante majeure du processus d'entraînement. Pour avoir le maximum d'effet, du personnel devrait être assigné immédiatement pour les projets Ada où des ingénieurs expérimentés dans la technologie Ada devraient être disponible pour consultation pour les premiers 4 à 6 mois. Des sources d'information pour des cours sont disponibles chez les Ada Information Clearing House et la revue Ada-Jovial Newsletter.

4. ÉVALUATION DU COMPILATEUR ET SÉLECTION

Utiliser un tout nouveau langage de programmation comporte des risques. Le processus de sélection d'un compilateur doit identifier les critères-clés et tester les systèmes de compilation qui seraient candidats selon ces critères. Du moment qu'Ada a été proposé pour la portabilité et l'uniformité, on pourrait penser que tous les systèmes de compilation sont équivalents. Le fait est que les concepteurs d'Ada ont laissé une grande liberté aux implanteurs, afin que le langage puisse être adapté aux besoins des applications. Les coûts d'une bonne évaluation et sélection sont considérables, mais les coûts d'une évaluation inadéquate sont encore plus élevés et peuvent causer des résultats désastreux pour le système en développement.

4.1 Besoins

4.1.1 Besoins en matériel pour l'évaluation

Les ordinateurs d'exécution et de développement sont souvent choisis avant la sélection du compilateur, ce qui ne laisse pas de choix. Si d'autre part on

souhaite un changement révolutionnaire ou si il n'y a pas un compilateur pour un certain matériel alors d'autres options doivent être explorées. Le choix d'une architecture d'exécution est fréquemment déterminé par les standards militaires. Pour des systèmes de développement et d'exécution différents, le chemin de déchargement peut être un goulot d'étranglement. Si l'ordinateur de développement est en connexion avec l'ordinateur d'exécution par un lien sériel à 9600 bauds, le déchargement de 64 kilo-octets prendrait plus qu'une minute; pour un demi-méga-octets, le temps serait approximativement de 10 minutes. L'évaluateur doit considérer avec attention l'impact du temps de déchargement et chercher des alternatives au transfert sériel. L'équipement de test peut être une ressource matérielle pour les applications essentielles dans le temps. Un avantage est que l'équipement de test matériel ne perturbe pas l'environnement. Le logiciel peut être exécuté exactement de la même façon si l'information de temporisation est collectée ou non. Pour des applications distribuées, il est important d'avoir la capacité de corréler les informations temporelles provenant de plusieurs sources indépendantes.

4.1.2 Besoins en logiciel pour l'évaluation

Le système de compilation inclut normalement les composantes logicielles suivantes: un compilateur, une bibliothèque de gestion, un système d'exécution, un éditeur de liens, un chargeur ou un déchargeur, un "debugger", un assembleur et un simulateur. Les cinq premiers éléments sont absolument nécessaires pour exécuter des programmes Ada. Le "debugger" est hautement souhaitable, et l'assembleur et le simulateur peuvent l'être aussi pour certaines applications spécialisées intégrées à l'équipement. Un APSE peut contenir des outils additionnels, parmi lesquels des outils de conception graphique, des analyseurs statiques et dynamiques, des outils de tests, des programmes d'impression avec mise en forme, etc. Les outils mentionnés coopèrent avec les outils de l'environnement de programmation. L'usager pourra peut être rencontrer un problème lors de la vérification de l'orthogonalité entre les outils de la bibliothèque Ada et les outils de l'environnement de programmation. Il pourrait alors avoir lieu de modifier une partie de l'environnement d'exécution.

4.1.3 Besoins de l'ensemble de tests

Pour obtenir un ensemble de tests, le coût est souvent négligeable comparé à celui de la préparation et de l'exécution de l'ensemble de tests et de l'interprétation exacte des résultats. C'est pour cela qu'il est important d'avoir des objectifs de tests spécifiques avant de commencer. La première décision qui s'impose porte sur le choix entre: acquisition ou construction. Si l'acquisition a été décidée, il faut sélectionner l'ensemble de tests, et à l'intérieur de cet ensemble, les tests spécifiques à exécuter. Si une évaluation à court terme pas trop sophistiquée est envisagée les tests PIWG devraient être suffisants. Si une évaluation plus approfondie est désirée, les tests ACEC ou AES devraient être plus adaptés.

4.2 Échéancier, cartes de temps, dépendances et coûts

On doit mettre tout en oeuvre pour s'assurer qu'un calendrier et un budget suffisants sont alloués à l'évaluation.

L'échéancier et le budget pour le processus de planification dépendent fortement des niveaux d'efforts alloués au processus d'évaluation. En général, on peut dire qu'il est prudent d'accorder 25% de l'effort total aux étapes de planification. Le temps d'exécution de l'évaluation est hautement variable, très peu pourra être accompli en moins de 2 semaines d'effort. Il est plus réaliste de considérer une période d'environ deux mois pour exécuter les tests et contacter les vendeurs et les utilisateurs. On apprend certainement sur la technologie d'exécution des tests en les exécutant mais il n'y a pas de gain significatif après la deuxième évaluation. L'analyse des résultats peut demander des ressources de temps importantes mais peut être facilité par du logiciel approprié. Par exemple, l'ensemble de tests ACEC comporte des programmes pour la comparaison des résultats de l'évaluation de performance entre plusieurs systèmes de compilation. Finalement, si les données ont été bien présentées et analysées, la sélection prend un effort minimal. Toutefois l'acquisition, l'installation, la vérification et l'acceptation peuvent impliquer une activité de plusieurs mois. Une stratégie qui a été efficace dans certains cas est de retarder la sélection finale d'un système de compilation jusqu'à ce

que la conception au niveau plus élevé soit presque complète. En plus de permettre au compilateur d'évoluer, cette stratégie fournit aussi du temps supplémentaire pour développer et exécuter sur le système candidat certains tests de prototypes ou d'autres tests spécifiques au projet. Enfin, l'impact du calendrier et du budget de l'évaluation et de la sélection d'un compilateur dépend fortement de la situation. Il faut compter de 1 à 6 mois ou plus pour le compléter avec 1 à 3 personnes. Les coûts peuvent s'élever à des dizaines de milliers de dollars si les frais de personnel sont calculés. Soulignons l'importance des compromis entre le coût d'une évaluation complète et le coût à long terme d'un projet pour lequel un système de compilation inapproprié ou moins désirable a été choisi.

4.3 Évaluation des vendeurs

Les indications pour l'évaluation de vendeurs des systèmes de compilation Ada ne sont pas très différentes de celles utilisées pour évaluer les vendeurs d'autres produits matériels et logiciels. La santé financière, le service et le support au produit sont importants, ainsi que la réaction du vendeur aux rapports d'erreurs et aux requêtes d'adaptation du compilateur. Les aspects suivants devraient être considérés:

- Structure corporative. Le système de compilation doit être produit par une seule compagnie, sinon comment les problèmes sont-ils reportés et résolus?

- Performance de la corporation. Est-ce-que le vendeur réagit positivement aux requêtes pour information?

- Ligne de produit.

- Santé corporative.

- Politique d'adaptation. Est-ce-que le vendeur est en mesure ou disponible pour adapter le compilateur pour les besoins des applications spécifiques?

- Politique de support.

- Politique de prix.

- Droit d'exécution. Est-ce-que le vendeur charge un droit pour chaque copie d'un programme d'application? Cette royauté pourrait être la composante la plus coûteuse pour une application qui est dupliquée à un millier d'exemplaires.

- Code source. Il est important de vérifier si le code source est disponible pour le compilateur du système d'exécution, avec les conditions et les coûts auxquels ils sont disponibles. Autrement il faut vérifier si les codes sources peuvent être mis en dépôt en tierce main de sorte que si la compagnie ferme, le développeur du système d'application ait une chance de résoudre les problèmes et corriger les erreurs.

- Aspect contractuel.

- Référence. Il faut vérifier si le système de compilation a été utilisé pour développer du système de logiciel similaire à l'application considérée et si le vendeur est disponible pour fournir ces références. L'acheteur devrait insister pour parler à des références techniques, et non seulement à des gestionnaires de projets.

4.4 Procédures d'évaluation

De façon générale, le processus d'évaluation indépendante peut prendre d'un à six mois pour une évaluation raisonnable la première fois. La durée dépend de plusieurs facteurs; en général, il est plus facile d'évaluer les systèmes de développement que les systèmes d'exécution. Évaluer un compilateur pour un système avec lequel on est déjà familier est plus facile qu'évaluer un nouveau système. Évaluer un compilateur est plus facile quand les

évaluateurs sont expérimentés. C'est plus facile si les compilateurs qui sont évalués sont matures et stables que s'ils sont des produits nouveaux basés sur des composantes nouvelles, ou si on est en train d'évaluer des compilateurs pour un seul ordinateur d'exécution. Même si le compilateur pour un programme a déjà été spécifié, il est toujours nécessaire de compléter une évaluation rigoureuse du compilateur. Si celui-ci ne répond pas aux besoins minimaux pour le programme, alors des conceptions alternatives de systèmes, un nouveau compilateur ou des changements de processeur ou de langage doivent être explorés. Les conséquences d'une sélection pauvre du compilateur Ada pour une application particulière sont des coûts potentiellement beaucoup plus élevés qu'une évaluation complète. Par exemple, des systèmes d'exécution qui sont inefficaces ou qui portent aux erreurs rendent le travail particulièrement difficile et demandent une grande qualité de support de la part du vendeur. Des fonctionnalités Ada standard qui ne sont pas supportées peuvent rendent impossible la réalisation des besoins fonctionnels. La dégradation de performance sous charge peut aussi rendre impossible la réalisation des besoins de performance, et changer le compilateur au milieu du projet s'avère coûteux. Le support du vendeur après l'acquisition est normalement très important. Des compilateurs relativement nouveaux fonctionnent rarement sans problème et la vitesse de réponse du vendeur pour la résolution de ces problèmes peut mener un projet au succès ou à l'échec. Les problèmes de capacité sont souvent difficiles à corriger ou à éviter. Des erreurs dans les systèmes d'exécution ne sont pas évidentes jusqu'à ce que le système ait été en opération pour une longue période de temps. Par exemple, si le système d'exécution n'alloue pas et ne libère pas correctement la mémoire pour des exceptions, l'erreur ne peut pas être détectée jusqu'à ce qu'un grand nombre d'exceptions aient été traitées. Des stratégies de réduction de risque pour cette situation incluent un test du système très tôt dans le cycle de développement et une stricte relation de travail avec le vendeur du compilateur.

4.5 La technologie du compilateur

Les évaluateurs sont avisés de tester les caractéristiques qui sont requises par leur application. La gestion des interruptions est très importante pour la

plupart des systèmes et en particulier pour ceux spécialisés intégrés aux équipements. Cette gestion des interruptions dépend de l'architecture de l'ordinateur d'exécution, du vendeur du compilateur, de l'application et du programmeur. Le développeur devrait être intéressé non seulement au produit final mais aussi au processus de développement et d'entretien. C'est pour cela que même si un compilateur génère du code fiable qui est vite exécuté, on doit quand même consacrer du temps à évaluer le système de compilation selon des critères additionnels. Par exemple, l'expansion du code est une mesure qui indique le montant d'espace qui sera requis par l'application. La capacité que possède l'éditeur de liens d'exclure des parties du code qui ne sont pas utilisées est une caractéristique à considérer. Un autre aspect très important est la vitesse à laquelle le temps de compilation et l'édition de lien croît avec la taille du programme. Certains compilateurs peuvent se comporter adéquatement pour des petits programmes, mais ils se dégradent sévèrement si la taille du programme augmente. Il faut aussi vérifier si le système de compilation évite de recompiler des unités pour lesquelles cette opération n'est pas nécessairement requise.

Actuellement, on connaît très peu d'évaluations de produits spécifiques dans le domaine public, pour plusieurs raisons. D'abord, ces études sont dépassées à peu près au moment même de leur publication, parce que les compilateurs sont en évolution et en amélioration constante. Deuxièmement, elles doivent être documentées avec soin et défendues contre les attaques des vendeurs. Ces vendeurs mettent leur produit sous la meilleure perspective possible, ils choisissent naturellement des paramètres favorables aux compilateurs pour la compilation, l'édition des liens et le chargement de leur programme, quand ces paramètres-là ne sont pas spécifiés explicitement ou éliminés dans les instructions pour l'exécution des programmes de référence.

Des exceptions remarquables au manque d'évaluation sont les évaluations des compilateurs basés sur le Ada Evaluation Systems que le British Standards Institute est en train de publier. Des ensembles de tests universels qui sont couramment disponibles sont le ACEC, le AES et le PIWG. Des tests spécifiques à certaines applications sont disponibles dans le "Dépôt de logiciels Ada" (Ada Software Repository).

La qualité de ces ensembles de tests est adéquate seulement s'ils sont utilisés par une personne avertie qui est familière avec les pièges de l'évaluation des performances. Les tests ne vérifient pas nécessairement les caractéristiques dans la même combinaison ou sous les mêmes conditions que celles employées dans une application particulière. Le meilleur logiciel à utiliser pour des comparaisons de systèmes de compilation est le logiciel opérationnel lui-même.

Pour les applications en temps réel, des critères importants d'évaluation sont: un comportement déterministe, la modélisation du temps, la gestion des interruptions, les clauses de représentation, les caractéristiques dépendantes de l'implantation et l'efficacité d'exécution en temps et en espace. Si les applications utilisent les tâches Ada, alors les caractéristiques de synchronisation seront importantes.

4.5.1 Évaluation d'options du compilateur et caractéristiques spéciales

Le standard ANSI/MIL-STD-1815A définit la syntaxe et la sémantique du langage Ada mais le vendeur du compilateur a beaucoup de choix pour fournir des options des caractéristiques qui ne sont pas requises par le standard. Il existe deux façons d'effectuer cela: la première utilise une directive à l'environnement de programmation quand le compilateur est appelé, et la deuxième opte pour une construction de langage appelé "pragma". Il est possible que les pragmas et les directives au compilateur soient en conflit entre elles. Même si le manuel de référence affirme que les pragmas définis par le langage doivent être supportés par chaque implantation, il s'avère que le niveau de support varie d'une implantation à l'autre.

À la différence des pragmas, les attributs doivent avoir l'effet décrit dans les manuels de référence. En outre, l'implantation peut fournir des attributs définis par l'implantation pourvu que l'indicateur de l'attribut ne soit pas le même qu'aucun autre attribut défini dans le langage. Les attributs définis par l'implantation doivent être décrits dans l'annexe F du manuel de référence.

4.5.2 Compilation incrémentielle

La compilation d'une unité quelconque dans la librairie des programmes peut rendre invalide les unités qui en dépendent. Par exemple, si les spécifications d'un sous-programme sont changées, toutes les unités qui utilisent cette définition sont sujettes à recompilation. Alors, il est extrêmement important de déterminer la granularité à laquelle la recompilation est requise. La solution directe et brute est simplement de marquer l'unité comme ayant besoin de compilation sans se soucier des impacts. Une approche plus sophistiquée est requise pour déterminer le fragment le plus petit du programme qui a besoin d'une recompilation. La compilation incrémentielle est l'habileté à utiliser des informations sur des compilations précédentes pour exécuter de nouvelles compilations à un coût inférieur. Si les modifications du fichier source sont dans les commentaires, la recompilation est une opération banale. Il faut noter que la compilation sélective ou la compilation incrémentielle ne sont généralement fournies par les compilateurs d'aujourd'hui. De bonnes représentations intermédiaires comme les arbres DIANA sont nécessaires pour capturer et réutiliser des informations provenant des compilations précédentes.

4.5.3 Efficacité en temps du code généré

L'efficacité du code généré par le compilateur peut être vérifiée par inspection et test. L'inspection du code généré par des programmes simples fournit une évidence historique de la qualité et de l'efficacité du code. Beaucoup d'informations peuvent être gagnées à partir d'un petit ensemble d'exemples soigneusement choisis. Des programmes de référence, d'un autre côté, peuvent être utilisés dans une sorte de boîte noire pour des tests plus exhaustifs et automatisés. De nombreuses optimisations sont souvent adoptées pour améliorer l'efficacité du code généré. Les procédures de tests donnent le résultat de l'efficacité du code généré mais n'indiquent pas la cause de ces résultats; l'inspection, par contre, fournit cette réponse.

4.5.4 Efficacité d'espace du code généré

Le montant de mémoire demandé par un programme Ada dépend du code-objet généré par le compilateur et par le système d'exécution. L'habileté à configurer le système d'exécution est l'un des besoins les plus importants pour des applications avec des contraintes d'espace. Pour l'évaluateur, les besoins d'espace sont probablement mieux mesurés à partir d'une application de référence représentative.

4.6 Critères quantitatifs et évaluation des performances

Le E & V Reference Manual [WRIG88a] et le E & V Guidebook [WRIG88b] est une référence globale pour les critères qui peuvent être utilisés afin d'évaluer du logiciel en général et des APSE en particulier. Les tests qui mesurent les caractéristiques de performance et d'efficacité d'un compilateur Ada sont souvent nommés "benchmarks" parce qu'ils sont un moyen pour comparer facilement un système à un autre. Un exemple de critère de performance est l'efficacité du temps de compilation. Similairement, des tests peuvent être écrits pour mesurer le temps nécessaire pour effectuer l'édition de liens des modules compilés indépendamment et l'efficacité en terme de temps et d'espace du code généré par le compilateur. Il est désirable de connaître le temps d'exécution d'une certaine instruction, le temps d'allocation d'une routine avec un certain nombre de paramètres, l'espace nécessaire pour implanter certaines instructions Ada et certaines structures de données comme les enregistrements. Des tests de capacité peuvent être construits pour déterminer la taille maximale des programmes qui peuvent être traités par un compilateur ou pour établir la taille de la table de symboles qui peut être traitée dans une certaine configuration de systèmes.

4.7 Informations complémentaires

Les évaluateurs ne doivent pas seulement se fier à l'information générée à l'interne; d'autres informations sont disponibles ailleurs. Les rapports de validation sont facilement disponibles; les vendeurs ont aussi des informations sur leur produit, mais ils les présentent toujours selon la perspective la plus

favorable. La source extérieure d'information s'averant peut-être la plus utile provient des utilisateurs du produit. Aucune évaluation ne devrait être considérée complète sans des entrevues intensives avec des usagers courants.

4.8 Réévaluation

La réévaluation du système de compilation peut être souhaitable dans trois circonstances: le besoin de mettre à jour le compilateur pour une nouvelle version, de changer l'ordinateur de développement ou d'exécution.

4.9 Évaluation d'adaptation

Il est très peu probable que l'évaluation d'un compilateur effectuée pour un projet réponde aux besoins d'un autre projet. Deux applications différentes ont des besoins différents pour un système de compilation Ada, et ce qui est optimal pour l'un ne l'est pas nécessairement pour l'autre. Par exemple, on remarque des besoins différents entre les systèmes en temps réel, les applications spécialisées intégrées à l'équipement et les systèmes de gestion de l'information qui ne sont pas en temps réel (MIS). D'autres applications en temps réel doux, en C^3I (commande, contrôle, communication, information) ou pour l'éducation ont des besoins différents. Inévitablement, les usagers adapteront leur style de programmation aux caractéristiques du compilateur Ada parce qu'il n'existe peut-être aucun système de compilation qui réponde à tous les besoins. Il est important que les besoins déterminent les compilateurs plutôt que l'inverse.

Chaque application devrait avoir ses propres besoins et critères d'évaluation. On connaît rarement a priori tous les besoins fonctionnels de performance et de support d'un système de compilation. Il est préférable de baser une sélection sur des critères de haut niveau plutôt que sur des critères de bas niveau, parce que l'interaction mutuelle des différentes caractéristiques est inconnue. Par exemple, il vaut mieux spécifier pour un système de gestion d'information le temps d'exécution d'un certain programme de traitement de fichiers plutôt que spécifier le temps d'accès aux enregistrements du fichier.

5. ÉVALUATION DE PERFORMANCE

5.1 Performance de compilation et d'édition de liens

Le montant de temps et d'espace disque requis pour la compilation et l'édition de liens des programmes Ada est important pour des projets Ada de très grande envergure. La distinction entre la compilation et l'édition de liens en Ada est souvent floue car certains vendeurs retardent des opérations comme l'instanciation des génériques jusqu'à l'édition de liens, alors que d'autres exécutent certaines opérations d'édition de liens à l'intérieur de la compilation. Malheureusement, l'évaluation des standards utilisés n'est pas satisfaisante pour la comparaison de performance. Typiquement, un vendeur de compilateur mentionnera seulement des chiffres qui donnent la vitesse du compilateur en terme de lignes de code compilées par minute. Très rarement, il mentionne les besoins de disque et de mémoire pour la compilation.

Parmi les facteurs qui influencent les mesures de lignes de code par minute, citons (la configuration, le matériel et le système d'exploitation devraient toujours être déclarés):

- La définition claire d'une ligne de code. Normalement, on peut considérer les retours du chariot ou le nombre de points-virgule.

- La taille d'un programme. Il est possible que des petits programmes soient compilés plus lentement et qu'ils génèrent plus de codes par ligne.

- Le nombre de commentaires et de lignes blanches si les retours du chariot sont utilisés comme ligne.

- Le type d'instruction.

- Les options du compilateur sélectionné. L'optimisateur et le "debugger" ralentissent généralement le processus de compilation.

- Le nombre de sous-unités et de génériques. Ces derniers en particulier peuvent avoir un grand impact sur la performance du compilateur.

- La définition de temps.

- Les paramètres du système d'exploitation, par exemple la taille de la partie active d'un programme.

- La taille et l'état de la bibliothèque du programme, par exemple concernant les problèmes de la fragmentation.

Parmi les questions qui doivent être soulignées, notons les suivantes:

- Comment le temps de compilation varie-t-il avec la taille d'un programme, sa complexité, les dépendances d'unité et les optimisations?

- Est-ce qu'un compilateur peut être utilisé simultanément par plusieurs utilisateurs?

- Comment le temps de compilation varie-t-il avec la charge du système ou le nombre d'usagers?

- Comment les besoins d'espace disque et de mémoire varient-ils avec la taille d'un programme, sa complexité et d'autres facteurs?

- Est-ce que le système de compilation fait le ménage après l'utilisation temporaire des fichiers sur disque?

- Combien d'espace est demandé pour tous les fichiers dérivés du fichier source pendant la compilation et jusqu'à qu'elle mesure leur taille peut être contrôlée?

- Est-ce que le système fournit les deux types de compilation interactive? Si oui, comment se différencient-elles?

- Est-il possible d'éviter des recompilations non nécessaires ou d'avoir de la compilation incrémentielle?

6. ÉVALUATION DE L'INTERACTION PERSONNE-MACHINE

6.1 Sorties d'information et diagnostics

Le système de compilation fournit normalement les options suivantes:

- Liste imprimée du compilateur.

- Liste imprimée de l'assembleur.

- Rapport d'erreurs.

- Avertissements.

- Aide interactive.

- Récupération des erreurs.

Un montant maximal d'information devrait être dérivé après chaque compilation, afin de favoriser la récupération des erreurs et le traitement du plus grand nombre possible d'unités de compilation.

6.2 Documentation

La documentation sur papier et en mode interactif est indispensable pour l'utilisation et la compréhension d'un système. Certaines caractéristiques d'une bonne documentation sont les suivantes:

- Contenu.
- Organisation.
- Style.
- Dépendance des implantations. Ces documents devraient utiliser des couleurs, des annotations marginales ou d'autres techniques pour distinguer clairement les caractéristiques qui dépendent de l'implantation.
- Annexe F - Après l'index, c'est la partie la plus importante de la documentation. Elle devrait être complète et dans l'ordre spécifié par le manuel de référence.
- Capacités et limitations.
- Options d'implantation.
- Messages d'erreurs
- Instructions d'installation.

7. ÉVALUATION DES FACTEURS ENVIRONNEMENTAUX

7.1 Technologie de l'environnement

7.1.1 Outils du APSE

Les besoins d'outils pour le MAPSE incluent: des éditeurs de texte, des programmes d'impression avec mise en forme, des analyseurs statiques de flux de contrôle, des outils d'analyse dynamique, des routines d'interface de terminaux, etc. D'autres outils, qui feront alors partie d'un APSE, peuvent être importants dans la définition des besoins et dans les phases de tests du cycle de vie. Parmi ceux-ci, on trouve des systèmes de traitement de la documentation, des vérificateurs d'orthographe, des aides à la planification, des outils de graphisme, etc.

L'existence d'un ensemble d'outils complets et intégrés peut faire partie des critères pour la sélection du compilateur. Il a certainement un impact sur la productivité du programmeur. Un autre critère désirable pour un environnement est l'ouverture, c'est-à-dire l'habileté à ajouter des outils construits ou achetés. Ce critère est souvent en conflit avec l'intégration étroite.

7.1.2 Bibliothèque de système

Les fonctions appartenant à la bibliothèque de système supportent la compilation séparée des unités de programme, ce qui facilite la programmation descendante et ascendante. Étant donné que les compilateurs Ada exécutent des contrôles multiples à travers des unités de compilation, la structure et la performance de la bibliothèque du système sont essentielles pour la performance et le temps de compilation. En Ada, la compilation indépendante ainsi que la connaissance de certaines propriétés définies dans d'autres modules sont disponibles aux modules en cours de compilation. Le but de l'évaluation de la bibliothèque de système est de déterminer comment les mécanismes de bibliothèque peuvent affecter la productivité du programmeur.

7.1.3 Support à l'éditeur de liens et au chargeur

L'éditeur de liens et le chargeur peuvent être fournis par le système d'exploitation; dans ce cas, les caractéristiques et la performance sont déterminées par le vendeur du système d'exploitation. Certains environnements sont intégrés tellement étroitement que le processus d'édition de liens est essentiellement invisible parce que les liens du programme sont édités incrémentiellement. Avec un système d'ordinateur de développement, l'éditeur de liens et le chargeur peuvent être couplés étroitement, mais dans un système de développement différent du système d'exécution, le chargeur nécessite une étape séparée et très dépendante de la machine d'exécution.

7.1.4 Chargement sélectif

Il n'est pas nécessaire que toutes les opérations fournies dans un paquetage soient utilisées. On obtient une meilleure efficacité si l'éditeur de liens ou le chargeur inclut dans le module chargé seulement les routines qui sont actuellement référées par le programme objet. Des considérations similaires s'appliquent au chargement sélectif de seulement ces modules dans le système d'exécution qui sont référées par le programme objet. L'absence des caractéristiques d'édition de liens sélective porte à créer des modules exécutables de centaines de milliers d'octets, quand seulement quelques dizaines de milliers d'objets sont peut-être réellement requis.

7.1.5 Le modèle du système d'exécution

Trois modèles d'exécution peuvent être définis pour des programmes Ada. D'abord, l'environnement basé sur l'ordinateur de développement dans lequel le code Ada peut utiliser les services du système d'exploitation. Deuxièmement, l'environnement des machines nues dans lesquelles le logiciel d'exécution est fourni par le système de compilation. En dernier lieu, l'environnement de machines nues augmentées dans lesquelles certains services d'exécution peuvent être fournis par un superviseur indépendant. Deux composantes doivent être considérées dans l'évaluation de la performance d'un système d'exécution d'un système de compilation Ada: le

code qui est généré à partir de programmes d'application, et le code qui fournit l'environnement dans lequel le programme sera exécuté. Le système d'exécution Ada doit fournir certains services parmi lesquels: la gestion de la mémoire, la gestion des tâches, la gestion du temps, le traitement des exceptions et le traitement des entrées/sorties. Ces services doivent être beaucoup plus économiques en terme de performance de temps et d'espace que le code généré par le programme d'applications. Ces fonctions peuvent être implantées comme du code généré ou comme appel à une fonction du système. Le choix entre les deux techniques a un impact sur la vitesse des opérations ainsi que sur la taille du système d'exécution. Des compromis sont nécessaires entre la taille du système d'exécution et la taille et l'efficacité du code généré, et il est très important de les reconnaître. Ordinairement, les compilateurs basés sur l'ordinateur de développement utilisent beaucoup les services fournis par le système d'exploitation. Le vendeur d'un compilateur en général n'a pas d'autre choix que d'utiliser les services du système d'exploitation. Similairement, l'évaluateur n'a souvent pas d'autre choix que d'évaluer le système de compilation dans le contexte du système d'exploitation. Les compilateurs pour une machine d'exécution ont deux alternatives: le système de compilation peut fournir la totalité du système d'exécution ou le système d'exécution peut être fourni par un autre superviseur. Les avantages des implantations pour une machine nue sont que le vendeur a le contrôle complet et peut fournir des systèmes hautement optimisés. D'un autre côté, le vendeur peut avoir peu d'expérience dans l'implantation de certaines fonctions du superviseur et vend alors un produit qui a moins de maturité.

7.1.6 Caractéristiques du système d'exécution

Des compromis sont inévitables entre la généralité et la performance pour des applications en temps réel ou spécialisées et intégrées à l'équipement. Il peut être nécessaire de sacrifier la généralité au bénéfice d'une performance accrue.

7.1.6.1 Efficacité de temps

L'efficacité du système d'exécution peut être plus importante que l'efficacité du code généré puisque l'usager a moins de contrôle sur la première. Les aspects à considérer sont, notamment, les tâches, le traitement des exceptions, les entrées/sorties.

7.1.6.2 Efficacité d'espace

De l'espace en mémoire est requis pour le code du système d'exécution (espace qui est déterminé au moment du chargement) pour les entités créés implicitement et explicitement. L'évaluateur doit se sentir concerné par le montant d'espace requis, la méthode d'allocation et de libération, et la réutilisation de l'espace non utilisé. Les besoins en mémoire sont particulièrement importants dans les systèmes spécialisés intégrés aux équipements qui n'ont pas de capacité de mémoire virtuelle parce que l'espace pris par le système d'exécution n'est pas disponible pour le code d'application. Un grand système d'exécution peut affecter la taille du programme à un degré plus élevé que la taille du code généré. La manière la plus directe et brute de construire un programme exécutable est d'inclure tout le système d'exécution. Si des grandes parties du système d'exécution ne sont pas utilisées, alors il s'agit d'un gaspillage d'espace. Un chargeur sélectif fourni par le vendeur est capable d'inclure ces parties du système d'exécution qui sont utilisées et d'exclure les parties qui ne le sont pas. L'évaluateur doit aussi déterminer si l'implantation exécute la récupération de la mémoire et, dans l'affirmative, sous quelles conditions. Le manuel de référence affirme qu'une implantation peut réclamer la mémoire occupée par un objet créé, une fois que cet objet est devenu inaccessible. La récupération de mémoire peut être faite dynamiquement (au moment de la libération d'un objet), périodiquement, ou quand il ne reste plus d'espace libre. Des applications en temps réel tolèrent rarement la récupération de mémoire à des temps imprévisibles. L'évaluateur doit donc déterminer l'impact de cette caractéristique en temps d'exécution.

7.1.7 Simulateur d'ordinateur d'exécution

La productivité du programmeur risque d'être diminuée à cause du temps requis pour le déchargement des programmes. Un simulateur d'ordinateur d'exécution permet l'exécution des programmes pour l'ordinateur d'exécution sur le système de développement et aussi de raccourcir le temps entre deux exécutions successives. Le simulateur devrait avoir exactement les mêmes caractéristiques que l'ordinateur d'exécution, par rapport à la vitesse d'exécution et à la synchronisation, à l'interface usager, aux fonctions de mise au point et aux interruptions.

7.1.8 Déchargement pour système d'exécution

Pour un système de développement différent de celui d'exécution, le chargement d'un programme Ada sur la machine d'exécution demande deux programmes: un déchargeur qui a été exécuté sur le système de développement, et un programme récepteur qui a été exécuté sur la machine d'exécution. Le processus de déchargement est souvent long, complexe et prédisposé aux erreurs. La performance de ce lien de communication peut avoir un impact significatif sur l'habileté à évaluer un système de compilation en exécutant des programmes de référence et, plus tard, sur le développement des systèmes. En fait, le temps utilisé pour décharger un programme peut dans certains cas être plus long que le temps de compilation, d'édition de liens et d'exécution combinés ensemble. Le temps de déchargement peut aussi augmenter remarquablement quand il faut décharger de l'information supplémentaire, comme la table de symboles, dans le cas de l'utilisation de moniteurs ou de systèmes de développement à micro-processeurs. Le temps de chargement peut être diminué si la partie statique de l'image exécutable tel que le système d'exécution ne doit pas être rechargé pour chaque exécution du programme. Ce lien critique de déchargement dépend en premier de la vitesse de la connexion matérielle, et doit être considéré très attentivement par le développeur qui utilise des systèmes de développement différents des systèmes d'exécution.

7.1.9 Support pour la mise au point

La mise au point des programmes Ada n'est pas nécessairement différente de celle des programmes écrits dans d'autres langages. En Ada, il n'y a pas de tests de référence pour les "debugger", mais l'évaluateur doit quand même examiner les aspects suivants: l'existence d'opérations pour le contrôle du flux d'exécution, de fonctions pour examiner l'état d'un programme, des opérations pour changer l'état d'un programme et des opérations pour montrer des informations de mise au point.

7.1.10 Effet d'optimisation sur la mise au point

Les optimisateurs ont tendance à confondre le "debugger" parce que l'optimisateur peut changer l'ordre d'opération dans un programme aussi longtemps qu'il n'affecte pas la sémantique du programme. De cette façon, les modèles au niveau du code-source de la sémantique ne représentent pas la sémantique de l'exécution. D'une façon similaire, les valeurs des variables pourraient être différentes de celles qu'on attend en regardant le code-source parce que le code-objet peut maintenir temporairement des valeurs dans les registres de la machine plutôt que de les emmagasiner dans la mémoire centrale. En combinant soigneusement certaines options du compilateur, on peut résoudre ces problèmes, mais les solutions ne sont pas sans limitation et sans coût. Il est important pour l'évaluateur de savoir quelles options offertes par le vendeur ont été choisies et de comprendre les compromis effectués et leurs implications.

7.1.11 Mise au point sur l'ordinateur d'exécution

L'ordinateur de développement est toujours utilisé pour la mise au point à cause des ressources informatiques disponibles. Deux techniques sont utilisées pour la mise au point. La première nécessite un noyau "debugger" sur l'ordinateur d'exécution qui envoie de l'information brute au "debugger" sur le système de développement. La deuxième technique, moins perturbatrice, consiste à utiliser du matériel pour examiner le but de

l'ordinateur d'exécution et pour transmettre les informations au "debugger" de développement.

7.1.12 Dépendances de l'implantation

L'évaluateur devrait consulter le CRID (Catalog of Runtime Implementation Dependencies) et déterminer si quelques dépendances d'implantation sont critiques pour l'application envisagée. La définition de langage contient beaucoup d'espace où les implanteurs Ada sont libres de choisir comment implanter une ou plusieurs caractéristiques du langage, en autant que la caractéristique se conforme aux règles du langage. Des sources d'information sont le "Guide des implanteurs ACVC" et le CRID du ARTEWG, qui sont organisées par référence de section au manuel de référence, ce qui permet de trouver facilement les dépendances d'implantation auxquelles nous sommes intéressés.

7.1.13 Traitement des interruptions

Le traitement des interruptions dépend beaucoup de l'implantation. L'évaluateur doit comprendre comment les interruptions sont traitées dans une implantation particulière et déterminer si ela est satisfaisant pour l'application. La plus grande partie des solutions alternatives ne sont pas portables entre la machine d'exécution ou entre différents compilateurs pour la même machine d'exécution. La fonctionnalité et la performance des interruptions fournies par le système de compilation devrait en être évaluées avec attention. Étant donné que le traitement des interruptions dépend tellement de la machine, il n'y a pas de code portable pour tester le traitement des interruptions. C'est pour cela que les aspects de traitement des interruptions sont largement ignorés dans les ensembles de tests de référence.

7.1.14 L'horloge et les aspects de synchronisation

L'évaluateur doit déterminer comment le temps est représenté, quelle est la résolution de l'horloge et quels sont les coûts d'appel des services d'horloge. Des paquetages déjà définis "calendar" and "system" définissent plusieurs

types et fonctions qui concernent le temps, par exemple le type "TIME", la fonction "CLOCK" et le type "DURATION". Pour des applications en temps réel qui mesurent le temps en microsecondes et millisecondes, l'abstraction du temps devient extrêmement importante. Pour ces applications, l'évaluateur doit déterminer plusieurs caractéristiques incluant:

- La précision des types DURATION et TIME.
- La vitesse à laquelle l'horloge fonctionne.
- Le maximum et le minimum de temps entre l'expiration d'un délai et la réorganisation de l'activité qui a exécuté le délai.
- La relation entre l'horloge Ada et l'horloge "matérielle" du système.

Si les caractéristiques de synchronisation ne sont pas accessibles, alors il reste l'option de créer une interface pour donner accès à une horloge matérielle qui est fournie avec tous les ordinateurs intégrés à l'équipement. La précision de ces horloges est d'une microseconde ou plus.

8. ASPECTS D'ÉVALUATION DE PERFORMANCE

L'évaluation des performances demande une analyse soignée et attentive effectuée par des gens très habiles techniquement. L'acceptation d'une simple comparaison sans une compréhension des tests et de l'environnement de tests est très risquée. L'évaluation des performances représente l'exécution d'un ensemble de programmes de tests sur un système pour comparer sa performance avec d'autres systèmes. Un problème dont les utilisateurs de programmes de référence devraient toujours être au courant est qu'il est difficile, sinon impossible, d'isoler le système de compilation des autres composantes de l'environnement; ceci inclut le matériel et le logiciel et peut être difficile à contrôler. Une mauvaise évaluation de performance peut être plus dommageable que l'absence d'évaluation.

8.1 Stratégie pour l'évaluation des performances

L'évaluation des performances devrait être faite très sélectivement. Il faut choisir les références les plus saillantes pour une application particulière. La

plupart du temps, on conseille à l'usager d'augmenter les programmes de référence sélectionnés avec des maquettes d'applications qui sont les plus représentatives de l'application choisie que tous les autres programmes de références publiquement disponibles. Il doit être reconnu que l'évaluation des performances est seulement un des aspects de l'évaluation d'un système de compilation, et ne devrait pas être le seul critère.

8.2 Information sur les configurations standards d'évaluations de performance

L'information complète sur la configuration d'évaluation est indispensable. Si un seul détail manque, l'utilisateur des données d'évaluation des performances doit le demander. Si l'information ne peut pas être fournie, le consommateur doit traiter des résultats avec un scepticisme extrême. En bref, les fournisseurs des données d'évaluations des performances doivent fournir toutes les informations qui sont nécessaires pour reproduire exactement les mêmes résultats en utilisant la même configuration. Si cette condition n'est pas remplie, il est très risqué de faire des hypothèses sur la configuration d'évaluation. Les résultats d'évaluation de performance devraient être considérés suspects jusqu'à ce qu'ils soient vérifiés et documentés.

8.3 Types de tests

Les programmes de référence peuvent être des petits programmes qui mesurent singulièrement une caractéristique Ada (ex. appel à une fonction), ou bien ils peuvent être de grands programmes qui mesurent plusieurs caractéristiques Ada combinées ensemble. Les programmes de référence à résolution fine sont utilisés pour mettre en évidence la force et la faiblesse d'un compilateur Ada. Les programmes des références à grande résolution sont des indicateurs utiles de l'efficacité globale de l'interaction mutuelle de plusieurs caractéristiques.

8.4 Tests de caractéristiques de langage

Les tests de caractéristiques de langage veulent isoler une seule ou un petit nombre de caractéristiques du langage Ada. La majeure partie des programmes de tests se place dans cette catégorie. Un ensemble de tests possède des vérifications pour mesurer le temps système de l'appel de sous-routines avec un nombre variable de paramètres, de direction des paramètres (entrées/sorties) et du type de paramètres. D'autres tests existent pour les opérations arithmétiques, pour le temps-système d'une itération, pour le temps-système d'accès aux composantes d'un "record type" etc. Si une application utilise lourdement une caractéristique particulière ou un ensemble de caractéristiques, alors l'usager pourrait désirer connaître le compilateur qui se comporte le mieux sur un certain ensemble de caractéristiques. L'usager pourrait alors influencer le vendeur pour qu'il opère des petits changements dans le compilateur dans des domaines où il est moins performant comparé à d'autres compilateurs. Les tests de caractéristiques et de langage ont pour désavantage de ne pas vérifier adéquatement l'impact de l'utilisation de caractéristiques combinées entre elles.

8.5 Tests de capacité et de dégradation

Ces tests servent à mesurer la capacité d'un système (matériel et logiciel) et à déterminer comment un système réagit à une utilisation croissante. La capacité d'un système est déterminée en grande partie par la mémoire et les limites internes. A titre d'exemple, mentionnons le niveau de l'emboîtement dynamique et l'espace de mémoire dynamique. Les tests de dégradation mesurent les changements de performance lorsque de lourdes charges de compilation font augmenter la longueur des queues du système et diminuer la disponibilité en mémoire.

8.6 Programmes de référence composites

Les programmes de référence composites sont des programmes servant à tester plusieurs caractéristiques combinées entre elles. Ils peuvent être dépendants ou indépendants de l'application. Ils sont rarement sélectionnés

sur une base scientifique, mais ils ont tendance à être très disponibles, ce qui rend la comparaison facile. Ils auraient pour avantage de pouvoir tester une large section de capacité du langage, ce qui n'est souvent pas le cas pour les tests disponibles. Ainsi un premier désavantage est que si les tests de référence produisent un résultat pauvre relativement à d'autres systèmes, il est souvent difficile d'en déterminer la raison. Ce peut être une simple caractéristique du langage, une inefficacité uniforme ou la façon dont le système de compilation traite les caractéristiques utilisées en combinaison. Des tests à résolution fine sont nécessaires pour examiner ces aspects.

8.7 Tests de référence synthétique

L'idée générale d'un programme de référence synthétique est de développer un squelette d'une application dont les caractéristiques sont représentatives. Les deux programmes de référence synthétique les plus connus sont le Whetstone et le Dhrystone. Le Whetstone est construit sur la base de la fréquence statique et dynamique d'instruction de 949 programmes. Il est construit pour être représentatif du calcul scientifique et numérique et fortement orienté vers des opérations en virgule flottante. Il est amplement accepté comme un moyen pour comparer des architectures de langage ou des implantations de langage. Le Dhrystone est censé refléter les caractéristiques des langages modernes de programmation (record et pointeur) et être représentatif de la programmation de systèmes. Les programmes de référence synthétique peuvent avoir plusieurs des avantages des programmes de références composites. Ils testent les caractéristiques du langage dans un sens ample et ils ont été construits scientifiquement. L'idée est de fournir toutes les caractéristiques de plusieurs programmes dans un seul programme exactement dans les proportions selon lesquelles ils existent dans quelques applications. Les désavantages des programmes de référence synthétique font qu'ils sont rarement acceptés comme étant représentatifs d'un domaine de problèmes particuliers et ils ne permettent pas à l'usager d'isoler une source particulière d'inefficacité.

8.8 Tests spécifiques aux applications

Les programmes de référence les plus représentatifs pour n'importe quelle application sont ceux qui composent l'application elle-même. Toutefois, il est souvent difficile et même impossible d'évaluer la performance d'un système en temps réel parce que le logiciel n'est pas encore écrit. La meilleure alternative est d'utiliser un programme de référence qui a été construit spécifiquement pour être similaire au programme d'application. On a besoin d'un programme avec du calcul similaire, des caractéristiques d'entrées/sorties semblables, un nombre de tâches semblable et des interactions similaires entre les tâches. L'avantage de ce type de programmes de référence est qu'ils sont plus représentatifs du travail réel devant être exécuté que n'importe quel autre programme de références. Les désavantages sont que nous pouvons être obligé de les construire à partir de zéro, ce qui est plus difficile que d'utiliser des programmes de référence déjà écrits. En outre, ces programmes de référence comportent les mêmes désavantages que les programmes de référence synthétique, c'est-à-dire qu'ils ne peuvent pas identifier clairement les sources de problèmes de performance.

8.9 Facteurs

Les résultats d'évaluation de performance peuvent varier d'une façon significative selon le matériel et l'environnement logiciel dans lesquels ils sont exécutés. Les évaluateurs devraient être attentifs aux différentes sources de variation et ils devraient essayer de les contrôler. Ils devraient aussi gagner une appréciation pour l'ampleur des erreurs qui sont introduites par ces sources de variation. Les programmes de référence offrent souvent des résultats inconsistants ou inexplicables. Ces variations dans les résultats dépendent de l'environnement dans lequel les programmes de référence sont exécutés. Les programmes de référence à résolution fine sont d'habitude plus sensibles aux variations que les programmes de référence à résolution plus large. Parmi les sources de variation, on peut inclure la mémoire (accès à la mémoire de la part des périphériques ou d'autres processeurs), le processeur (traitement des interruptions, etc.), le système d'exploitation et d'exécution

(récupération de la mémoire, temps système, etc.), et le compilateur (optimisation, etc.).

8.10 Anomalies dans la synchronisation

La plupart des ensembles de programmes de référence en Ada se fient à des techniques logicielles pour la synchronisation. Le paquetage CALENDAR fournit une capacité de synchronisation dépendant de l'implantation qui doit être attentivement étudiée et comprise. La plupart des programmes de référence utilisent le paquetage CALENDAR plutôt qu'un mécanisme de synchronisation externe ou du système d'exploitation. Le premier problème possible pour l'évaluation des performances est le manque de précision de l'horloge de ce paquetage. Plusieurs implantations Ada ont une résolution de l'horloge de dix millisecondes ou plus. Cette résolution est insuffisante pour une évaluation fine des performances de la synchronisation. La solution à ces problèmes est d'utiliser une structure de test à "double boucle". Selon cette technique, le logiciel qui doit être mesuré est exécuté plusieurs fois dans une itération. Le temps système de l'itération est soustrait en mesurant le temps d'une itération de contrôle. Cette technique dépend lourdement de l'hypothèse selon laquelle l'instruction dans la boucle de tests prend exactement le même montant de temps que l'instruction dans la boucle de contrôle. Les variations décrites dans la section 8.9 peuvent invalider cette hypothèse.

8.11 Vérification de la synchronisation

Les techniques de synchronisation du logiciel ne sont pas toujours fiables. On connaît actuellement deux techniques pour vérifier la synchronisation. La première est d'utiliser une horloge matérielle de haute précision avec un paquetage Ada d'interface, et la deuxième est d'utiliser directement le matériel (par exemple un analyseur logique ou un émulateur de circuits pour prendre des mesures). Les horloges matérielles sont généralement au moins de trois ordres de grandeur plus précises que les horloges logicielles. Elles permettent d'estimer la distribution de temps requis pour exécuter une certaine construction du langage plutôt que seulement en calculer la moyenne. Les

analyseurs logiques et les systèmes de développement à micro-processeurs (MDS) avec des émulateurs de circuits fournissent des capacités qui sont précieuses aux programmeurs de systèmes en temps réel spécialisés et intégrés à l'équipement. Les analyseurs logiques permettent à l'utilisateur de mesurer certains temps qui sont impossibles à déterminer en utilisant des techniques logicielles. Par exemple, aucune technique logicielle ne peut déterminer l'intervalle de temps entre le signal d'interruption et le début de sa routine de service. Alors, pour la programmation des traitements d'interruptions, pour la programmation en temps réel et pour la mise au point, l'analyseur logique et le MDS deviennent des outils presque indispensables.

8.12 Analyse des données et rapport d'évaluation

Il est extrêmement utile d'avoir des programmes qui analysent et montrent les données d'une façon utile mais ces outils ne peuvent pas substituer une analyse d'ingénierie profonde. Il est particulièrement important de découvrir les points faibles et les points forts de chaque compilateur. Le ACEC fournit quelques unes de ces routines d'analyse. Chaque compilateur Ada offre quelque chose de différent et devrait être jugé sur la base de ses forces et de ses faiblesses plutôt que de l'évaluer contre un système rigide de critères. Par exemple, un système de compilation qui est très bien évalué sur la plupart des critères mais a seulement un défaut fatal peut être inacceptable si on le compare à un compilateur qui est évalué moins bien mais uniformément sur tous les critères. Les ensembles de logiciels d'analyse peuvent donner l'impression que l'évaluation est basée sur une comparaison entre objets de la même catégorie quand, en réalité, les systèmes de compilation qui sont comparés n'examinent pas les mêmes problèmes et ne devraient pas être jugés en utilisant la même mesure.

9. ENSEMBLE DE TESTS ET AUTRES TECHNOLOGIES DISPONIBLES

[51]

9.1 Le Ada Compiler Evaluation Capability (ACEC)

Force: couverture extensive des caractéristiques de langages et des outils d'analyse.
Faiblesse: nouveauté relative et manque de support.

9.2 Les programmes de références PIWG

Force: ample distribution et disponibilité.
Faiblesse: manque de support et documentation.

9.3 Le système d'évaluation Ada

Force: ample couverture des aspects du système de compilation incluant la liste de vérification (check list).
Faiblesse: coût et le manque d'outils d'analyse.

9.4 Évaluation des références en Ada de l'Université du Michigan

Force: mécanisme de synchronisation basé sur des principes théoriques profonds.
Faiblesse: pas de développement ultérieur, ni de support.

9.5 Références de l'Aerospace Corporation

Force: le seul ensemble de tests qui permette la comparaison entre différentes implantations de langages.
Faiblesse: disponibilité et support encore sous considération.

9.6 Dépôt logiciel Ada

Force: tout est disponible en ligne gratuitement
Faiblesse: la qualité des programmes et de documentation est très variable dépendamment de la source.

9.7 Autres sources

Grebyn Corporation, INFOAda Newsletter, Cutter Information Corporation, Ada Stragegies Newsletter Guidebook, etc. Parmi les compagnies Texas Instruments, Sperry and General Dynamics.

10. COMPARAISON ENTRE ADA ET LES AUTRES LANGAGES

Indépendamment du fait qu'un langage est élégant ou fonctionnel, c'est l'implantation réelle du langage qui déterminera le succès ou l'échec d'un projet. Parmi les aspects qu'il faut considérer pour la comparaison, notons les suivants:

- Contrôle en temps de compilation.
- Contrôle en temps d'exécution.
- Représentation des données.
- Option de compilateurs et du système d'exploitation.

10.1 Interactions d'Ada avec d'autres langages

Mélanger Ada avec d'autres langages est possible, mais une considération particulière doit être donnée aux aspects techniques et de gestion. D'un autre côté, la traduction automatique du code écrit dans d'autres langages devrait être approchée avec attention.

10.2 Interfaces du code Ada avec d'autres langages

Une interface efficace entre Ada et un autre langage exige l'habileté à appeler des sous-routines écrites dans d'autres langages. Ceci peut être fait si les compilateurs supportent le pragma "Interface" pour ces langages. L'interface des données entre les deux langages doit être spécifiée et contrôlée de la même façon entre le processeur et les bus. Un des problèmes qui peut apparaître c'est que des compilateurs pas suffisamment évolués pourraient charger le support d'exécution. La mesure dans laquelle un compilateur charge seulement la partie du support d'exécution nécessaire est un facteur important pour décider s'il est possible et efficace d'écrire seulement une partie du système en Ada. Seule une analyse détaillée peut fournir la réponse à ces questions.

En ce qui concerne la comparaison d'Ada avec d'autres langages, on peut comparer seulement une implantation particulière d'Ada avec une implantation particulière d'un autre langage. Pour rendre la comparaison correcte, il doit être compris qu'Ada fait plus de contrôle de consistance au moment de la compilation et de l'exécution que plusieurs autres langages. Ces compromis doivent être reconnus et acceptés. Selon beaucoup de points de vue, l'évalution de compilateurs Ada est la même que celle utilisée pour d'autres langages. La complexité du langage, le nombre d'implantations et l'incorporation de certaines fonctions du système d'exploitation (comme la gestion de la mémoire et la synchronisation) dans le système d'exécution rendent l'évaluation Ada plus difficile. Le système de bibliothèque de programmes Ada, qui réalise la compilation séparée avec le plein contrôle de consistance fournit une dimension à l'évaluation qui ne peut pas être relevée dans plusieurs autres langages.

10.3 Remplacement de la totalité du système

Remplacer un système seulement pour l'avoir en Ada n'est pas généralement recommandé. Toutefois, Ada devrait être le langage de remplacement si une des situations suivantes existe:

- Pauvre fiabilité totale du système.
- 35% ou plus de changement immédiat.
- Remplacement du matériel.

10.4 Traduction automatique

On pourrait être tenté de traduire automatiquement du logiciel déjà existant dans un autre langage en Ada, mais cette opération comporte des risques significatifs. Une analyse complète des aspects techniques, des coûts et de la durée de vie du logiciel devrait être faite avant de s'engager dans une approche de traduction automatique.

10.5 Langage de traduction

La traduction automatique d'un langage de programmation vers un autre devrait être approchée avec beaucoup d'attention même si un certain nombre de systèmes de traduction existent et que la traduction offre la possibilité d'obtenir rapidement des programmes exécutables. Il existe des problèmes significatifs et des aspects qui doivent être considérés avant de se lancer dans cette opération, notamment:

- Degré de traduction.
- Perte des bénéfices du génie logiciel.
- Routines spéciales où la traduction n'est pas directement applicable.
- Test.

A court terme le langage de traduction doit être vu comme une proposition risquée qui fournira des résultats de qualité inconnue. Concevoir ou coder de nouveau le système en Ada ou interfacer du code Ada écrit récemment avec l'implantation courante sont des approches bien préférables. Si une stratégie

de traduction est considérée, une analyse profonde des traducteurs commerciaux et du domaine public est fortement recommandée comme une partie de l'étude de faisabilité. Ces recherches devraient utiliser un ensemble robuste de cas de tests et de programmes de référence qui sont typiques de l'application particulière.

11. ORIENTATIONS PRATIQUES

11.1 Scénario d'utilisation

- Activités à faible risque:

Applications qui ne sont pas essentielles dans le temps et qui sont exécutées sur des machines avec des capacités de mémoire de moyenne à large. Ces applications incluent des outils logiciels, certaines simulations et des applications pour des missions basées à terre.

- Activités à risque moyen:

Grands projets de commande, contrôle, communication et information (C^3I). Applications essentielles dans le temps, basées à terre, développées immédiatement et exécutées sur des machines avec une capacité de mémoire moyenne à large. Applications essentielles dans le temps pour l'air ou l'espace qui seront engagées dans un développement à pleine échelle dans les prochains 2 à 4 ans.

- Activités à risque élevé:

Utilisation immédiate pour l'espace ou l'aviation, du développement du code pour des applications essentielles dans le temps, qui utilisent des processeurs spécialisés intégrés aux équipements pour les forces armées, à moins que les problèmes possibles et les retards intrinsèques dans l'évolution du générateur du code et du système d'exécution en parallèle avec le développement de l'application puissent être tolérés.

11.2 Stratégie de départ

La meilleure stratégie pour une compagnie qui a peu ou pas d'expérience dans le développement des systèmes Ada est la suivante:

- Développer et exécuter un plan à long terme basé sur la formation qui inclut le personnel de gestion, les spécialistes d'applications, les concepteurs, les ingénieurs du logiciel et le personnel de test. La formation des gestionnaires est absolument indispensable.

- Structurer le calendrier du développement de façon à contenir quelques activités Ada non essentielles qui puissent être accomplies très tôt dans le programme afin de former du personnel expérimenté en Ada. L'emphase devrait être mise sur les efforts de petites équipes avec des revisions sur le développement en cours, de la distribution d'information et des observations de gestion.

- Répartir le personnel expérimenté du petit projet Ada comme leaders dans le développement principal.

- Arranger le plan de développement du logiciel selon les résultats des activités Ada.

- Considérer de ramener dans l'équipe des experts avec de l'expérience en développement de systèmes similaires en Ada pour évaluer la conception du système logiciel et matériel et la stratégie d'implantation.

11.3 Compilateurs

Pour l'acquisition des compilateurs, certaines suggestions peuvent être faites:

- Les ressources de l'ordinateur de développement devraient être plus grandes que pour d'autres langages.
- Évaluer le compilateur.
- Vérifier les services du compilateur.
- Considérer les coûts de recompilation.
- Comparer les coûts d'une autre validation avec les avantages d'une nouvelle version.

11.4 Processus de sélection

- Recueillir des informations générales d'évaluation.
- Planifier une stratégie globale qui inclut le budget, le personnel et les cartes de temps.
- Identifier les besoins du projet.
- Établir des critères basés sur la nature du projet.
- Planifier des tactiques dans les programmes de référence, des entrevues et la recherche d'information.
- Construire ou trouver un banc d'essai d'évaluation.
- Exécuter les évaluations.
- Analyser les résultats.
- Sélectionner, se procurer, installer et accepter le système de compilation.
- Rapporter l'information au vendeur.

11.5 Réutilisation

À court terme:

- Partage du code.
- Progiciels commercialement disponibles.
- Méthodologie. La réutilisation peut devenir un objectif explicite d'un projet.

À long terme:

- Bibliothèque initiale. Une bibliothèque de composants réutilisables peut être développée et entretenue. Une qualité excellente de documentation pour ces composantes est nécessaire.

- Bibliothèque complètement intégrée. Avec un catalogue des pièces logicielles.

- Système de composition.

11.6 Systèmes en temps réel

Aujourd'hui, pour assurer l'utilisation avec succès d'Ada par un système en temps réel, il faut:

- Contrôler le compilateur proposé, afin qu'il supporte efficacement la gestion des interruptions; sinon, la gestion des interruptions, qui est une petite partie du code du système, devrait être faite en langage standard.

- Utiliser un nombre limité de tâches Ada dans le code.

BIBLIOGRAPHIE

[AdaT82] United States Department of Defense. *AdaTEC : The SIGPLAN Technical Committee on Ada : Reference Manual for the Ada Programming Language*. Baltimore, MD : ACM, July 1982. (Pagination multiple)

[APPR89] *Approved Ada Language Commentaries : Ada Letters*. New York : ACM, 1989. xi, 341 pp. (Édition spéciale de SIGAda, ACM Special Interest Group on Ada, vol. IX, no. 3, Spring 1989)

[APSE88] *Future APSE Workshop III : ACM SIGAda*. Boston, MA, 1-2 décembre 1988. (Notes de travail)

[CAIS89] *Introduction to CAIS : Common Ada Programming Support Environment (APSE) Interface Set*. [Washington : The United States Department of Defense], September 30, 1989. x, 53 pp. (Rapport technique: MIL-STD-1838A)

[FEIL88] Peter H. Feiler, Susan A. Dart and Grace Downey. *Evaluation of the Rational Environment*. Pittsburgh, PA : Carnegie Mellon University, Software Engineering Institute, July 1988. v, 80 pp. (Rapport technique: CMU/SEI-88-TR-15, ESD-TR-88-16)

[FORE87] John Foreman and John Goodenough. *Ada Adoption Handbook: A Program Manager's Guide : Version 1.0*. Pittsburgh, PA : Carnegie Mellon University, Software Engineering Institute, May 1987. iii, 82 pp. (Rapport technique: CMU/SEI-87-TR-9, ESD-TR-87-110)

[IEEE89] *IEEE Transactions on Software Engineering*, vol. 15, no 3 (March 1989).

[LUIS89] Lui Sha and John B. Goodenough. *Real-Time Scheduling Theory and Ada*. Pittsburgh, PA : Carnegie Mellon University, Software Engineering Institute, April 1989. v, 28 pp. (Rapport technique: CMU/SEI-89-TR-14, ESD-TR-89-22)

[SIGA89] *Ada : A Driving Factor for Software Quality and Productivity Improvement : Conference Program and Abstracts : ACM SIGAda Conference 1989 : Ottawa Meeting.* Ottawa : ACM SIGAda, August 8-11, 1989. vli pp.

[TRIA89a] *Tri-Ada '89 : Ada Technology in Context : Application, Development, and Deployment : Proceedings.* New York : ACM, 1989. xxvi, 670 pp. (Conférence tenue à: David L. Lawrence Convention Center, Pittsburgh, PA, October 23-26, 1989)

[TRIA89b] *Tri-Ada '89 : Ada Technology in Context: Application, Development, and Deployment : Tutorial : Compiler Evaluation and Selection.* [New York : ACM, 1989. 40 pp.] (Conférence tenue à: David L. Lawrence Convention Center, Pittsburgh, PA, October 23-26, 1989)

[UNIT89a] United States Department of Defense. Office of the Under Secretary of Defense for Acquisition. *Ada 9X Project Report : Ada 9X Project Plan.* Washington : United States Department of Defense, January 1989. x, 31 pp.

[UNIT89b] United States Department of Defense. Office of the Under Secretary of Defense for Acquisition. *Ada 9X Project Report : Requirements Development Plan.* Washington : United States Department of Defense, August 1989. vi, 19 pp.

[UNIT89c] United States Department of Defense. Office of the Under Secretary of Defense for Acquisition. *Ada 9X Project Report : Requirements Workshop.* Washington : United States Department of Defense, June 1989. x, 87 pp.

[UNIT89d] United States Department of Defense. Office of the Under Secretary of Defense for Acquisition. *Ada 9X Project Report : Revision Request Report.* Washington: United States Department of Defense, August 1989. (Pagination multiple)

[WEID89] Nelson H. Weiderman. *Ada Adoption Handbook : Compiler Evaluation and Selection : Version 1.0.* Pittsburgh, PA : Carnegie-Mellon University, Software Engineering Institute, March 1989. vii, 118 pp. (Technical Report: CMU/SEI-89-TR-13, ESD-TR-89-21)

[WRIG88a] Wright Research and Development Center. *E & V Reference : Manual : Version 1.1.* Wright Patterson AFB, OH : October 1988. (Rapport technique: TASC-TR-5234-3 : DTIC)

[WRIG88b] Wright Research and Development Center. *E & V Guidebook : Version 1.1.* Wright Patterson AFB, OH : August 1988. (Rapport technique: TASC-TR-5234-4 : DTIC)

ACRONYMES

ACEC	Ada Compiler Evaluation Capability
ACM	Association for Computing Machinery
ACVC	Ada Compiler Validation Capability
AES	Ada Evaluation System
AI	Artificial Intelligence
AJPO	Ada Joint Program Office
ANSI	American National Standards Institute
APSE	Ada Programming Support Environment
ARTEWG	Ada Runtime Environment Working Group
AVF	Ada Validation Facility
AVS	Ada Validation Suite
C^3I	Command, Control, Communications & Intelligence
CAATS	Canadian Automated Air Traffic Systems
CAIS	Common APSE Interface Set
CASE	Computer Aided Software Engineering
CLI	Common Language Interpreter
CPU	Central Processing Unit
CRID	Catalog of Runtime Implementation Dependencies
DIANA	Descriptive Intermediate Attribute Notation for Ada
DoD	Department of Defense
E & V	Evaluation and Validation
FAA	Federal Aviation Administration
HOOD	Hierarchical Object Oriented Design
ISO	International Standards Organization
MAPSE	Minimal Ada Programming Support Environment
MCCR	Mission Critical Computer Resources
MDS	Microprocessor Development System
MIS	Management Information System
MSS	Mobile Servicing System
NASA	National Aeronautic & Space Administration
NIST	National Institute for Standards and Technology
OTAN	Organisation du Traité de l'Atlantique Nord
PDL	Program Design Language

PIWG	Performance Issues Working Group
SDE	Software Development Environments
SEDL	Software Engineering Design Language
SEI	Software Engineering Institute
SIGAda	Special Interest Group on Ada
STARS	Software Technology for Adaptable for Reliable System
URG	Uniformity Rapporteur Group
VMS	Virtual Memory System
VSR	Validation Summary Report

ANNEXES

Centre de recherche informatique de Montréal

décembre 1990

Centre de recherche informatique de Montréal

décembre 1990

RAPPORT DE COLLOQUES

1 ACM SIGADA

Ce colloque s'est déroulé à Ottawa, du 8 au 11 août 1989, sous l'égide de: ADGA Group, Ottawa, Ont.; APG Canada Inc., Montréal, Qué.; le Groupe CGI, Montréal, Qué.; DY-4 Systems Inc., Nepean; MacDonald Dettwiler & Associates, Richmond, B.C.; Ogivar/Harris, Ottawa, Ont.; Paramax Electronics Inc., Montréal, Qué.; PRIOR Data Sciences Ltd., Kanata, Ont.; Systemhouse, Ottawa, Ont.

Au cours du colloque, plusieurs interventions ont souligné l'importance d'Ada dans l'augmentation de la productivité du logiciel, la création de logiciel plus fiable et plus discipliné et dans le processus de développement du logiciel en général. Plusieurs conférenciers ont montré qu'Ada n'est pas seulement un langage de programmation mais une technologie qui demande de la connaissance en génie logiciel pour l'apprécier et en tirer tout le bénéfice. La réutilisation fait aussi partie des techniques qui sont nécessaires pour être en mesure d'affronter les besoins des années 90 et pour augmenter la productivité.

En ce qui concerne la réutilisation de logiciels, différents aspects et problèmes ont été présentés; mais en particulier il a été question de l'aspect touchant la loi sur les droits d'auteurs. Les questions soulevées portent sur la propriété intellectuelle d'un logiciel réutilisé plusieurs fois, sur la responsabilité en cas de défaut et sur la responsabilité de corriger les erreurs. Les problèmes rencontrés proviennent du fait que la réutilisation du logiciel implique souvent le droit de reproduire, modifier, augmenter et entretenir du logiciel, le droit de traduire le logiciel d'un langage à un autre et le droit d'étendre ses droits à des tierces parties.

Plusieurs projets en Ada sont significatifs et valent la peine d'être mentionnés. Le projet 9X Ada est une révision du langage pour les années 1990. Il ne

s'agit pas d'une nouvelle conception (redesign). Plusieurs documents sur le projet 9X sont déjà rendus publics, notamment les rapports sur la requête de révision, le plan de développement des besoins et le plan du projet. Ces documents sont disponibles au CRIM. Les objectifs du projet 9X sont de:

- réviser le standard en ANSI/MIL-STD-1815A
- obtenir l'approbation des organismes de standards américain et internationaux, et celle du ministère de la Défense
- améliorer l'ensemble des programmes de test ACVC et ACEC
- recommander des politiques de transition
- développer des programmes d'éducation et d'entraînement pour Ada
- développer un plan d'entretien à long terme du langage.

STARS, un programme de recherche de technologie avancée du ministère de la Défense américain, permet d'obtenir des améliorations radicales dans la qualité du logiciel, et de contenir et réduire l'augmentation imprévisible du coût du logiciel pour des applications spécialisées intégrées à l'équipement, ou essentielles pour les missions. Un des objectifs de STARS est de démontrer que la technologie et le processus pour créer du logiciel indépendant de la machine et réutilisable suivent une approche de "logiciel d'abord", par opposition au choix "matériel d'abord" qui est l'approche courante. Un autre objectif de ce programme vise l'obtention d'un environnement indépendant de la machine et adaptable à plusieurs domaines d'applications de support pour du logiciel destiné à des missions militaires. Le ministère de la Défense désire encourager l'acceptation commerciale des concepts et des outils de l'environnement de STARS.

Le projet d'environnement de support logiciel pour le cycle de vie développé par le General Research Corporation en accord avec le Standard DoD-STD2167A est un autre projet important. Les exigences de ce projet incluent le support pour:

- la totalité du cycle de vie du logiciel
- le support de plusieurs langages
- une performance élevée obtenue avec une "database machine"

- une architecture ouverte et évolutive
- la possibilité d'utiliser des capacités VMS existantes
- des capacités de fenêtres sur des terminaux orientés vers les caractères.

La NASA est aussi profondément impliquée dans des projets Ada avec l'étude de la Station Spatiale.

Comme il a été souligné, une partie de la recherche en Ada est de la recherche en génie logiciel. Par conséquent, les outils CASE (Computer Aided Software Engineering) en Ada font l'objet de nombreuses études. La méthode HOOD et ses ensembles d'outils de support ont été présentés par APG Canada et Ipsys PLC Angleterre.

Gerald Karam, de l'Université de Carleton, travaille sur des projets de génie logiciel en Ada. Il a présenté un article sur la prochaine génération d'outils CASE en décrivant des "Machine Charts". Un langage pour décrire la conception en génie logiciel a été présenté par IBM. Ce langage se nomme SEDL et a été proposé pour la spécification et la conception des programmes Ada. Des méthodes de développement pour des implantations distribuées sur plusieurs systèmes et pour le développement orienté-objet ont aussi été présentées et l'importance des standards graphiques en relation Ada a été soulignée.

Quant aux applications gouvernementales, plusieurs projets ont été présentés: le "Rear Admiral" Boyle du ministère de la Défense Nationale a présenté les politiques Ada et le plan d'implantation du ministère de la Défense Nationale. Ce plan inclut l'étude de l'ampleur du domaine du logiciel pour le contrôle de procédé, ainsi que des politiques Ada qui s'appliquent au logiciel pour le contrôle de procédé. Le major Létourneau, de la Défense Nationale, a présenté le projet APSE, de l'OTAN, en décrivant les composantes qui sont développées dans les différentes nations. La contribution canadienne à ce projet est le CLI (Common Language Interpreter) APSE et un éditeur guidé par la syntaxe (SDE). Il y a d'autres projets au Canada, notamment le projet du nouvel appareil transporté par navire, le projet CAATS (Canadian Automated Air Traffic Systems) de Transport Canada et le système de service

mobile (MSS) qui est la contribution canadienne à la station spatiale de la NASA.

La dernière partie du colloque concernait l'enseignement et la formation en génie logiciel et en Ada. Au Canada, Monsieur Joseph Côté, du Conseil du Trésor, invite fortement les entreprises intéressées par Ada à le contacter pour faire connaître au gouvernement leurs opinions sur l'impact industriel de certaines normes avant qu'elles soient rendues obligatoires.

2 TRI-ADA

La conférence TRI-Ada s'est tenue à Pittsburgh, PA du 23-26 octobre, 1989 et on y a enregistré un nombre de participants beaucoup plus important. Les présentations étaient classées dans les catégories suivantes: les applications, le standard DoD-STD-2167A sur le cycle de développement des projets Ada, les environnements de programmation, les outils, les méthodes formelles, la réutilisation du logiciel, la méthodologie orientée-objet, la formation en Ada, les systèmes en temps réel et distribué, l'évaluation de la technique des compilateurs et les aspects de structure du langage.

Dans les applications, nous avons remarqué plusieurs applications militaires sur des ordinateurs spécialisés et intégrés à l'équipement, l'implantation d'un serveur pour un système de fenêtrage en Ada, différentes applications assimilables au C^3I, (Command, Control, Communication and Intelligence) et un environnement de support au logiciel après son déploiement.

Le sujet du standard DoD-STD-2167A a eu des implications, entre autres sur la documentation de la conception et sur l'approche incrémentielle aux tests du logiciel.

Plusieurs environnements et outils ont été présentés: l'environnement Protob, L'APSE de l'OTAN, des systèmes experts en Ada et un environnement de développement basé sur réseaux.

Deux méthodes formelles ont été introduites: la méthode Penelope, pour la vérification des systèmes Ada et une méthode pour la spécification algébrique des ordres partiels générés par des calculs concurrents en Ada.

Plusieurs chercheurs du Software Engineering Institute (SEI) ont présenté le résultat de leurs recherches sur la réutilisation du logiciel. Le SEI est principalement intéressé à la réutilisation du logiciel pour les ressources informatiques essentielles aux missions (MCCR). Ada est censé faciliter la réutilisation textuelle des logiciels. Dans le débat sur la réutilisation, plusieurs éléments intéressants ont été présentés; toutefois les arguments n'étaient pas nouveaux. La définition de métriques pour mesurer la réutilisabilité du logiciel ainsi que d'autres caractéristiques comme l'entretenabilité, la portabilité, l'utilisabilité, font encore l'objet de recherches. De plus le manque d'enthousiasme pour la réutilisation est considéré comme un phénomène important. Les programmeurs sont rénumérés indépendamment du fait qu'ils réutilisent ou non du logiciel. Pour des raisons psychologiques, ils préfèrent ne pas utiliser quelque chose bâti par quelqu'un d'autre, mais plutôt le rebâtir eux-mêmes. Du point de vue de la gestion, il est important pour un gestionnaire de diriger un grand nombre de personnes pour avoir un salaire plus élevé; la réutilisation réduirait donc son équipe. Il n'existe pas non plus de motivations économiques: d'habitude dans un contrat (même avec le gouvernement), celui qui paie pour le contrat ne veut pas avoir de déboursés ultérieurs pour rendre son produit réutilisable. En outre, il y a toujours les problèmes de la Loi sur les droits d'auteur, déjà mentionnés, au colloque ACM SIGAda, qui rendent difficile la réutilisation du logiciel.

Des méthodes pour la représentation de la réutilisation du logiciel ont été présentées. La réutilisation a également été étudiée en fonction de son impact sur la définition du processus du développement du logiciel.

L'intérêt de la programmation orientée-objet pour le processus de développement Ada a été souligné par un certain nombre d'articles touchant les phases de programmation et de conception orientée-objet.

Dans la catégorie des applications en temps réel et distribué, les présentations portaient sur l'application de modèles des tâches Ada à l'affectation de la

CPU, les systèmes d'Ada distribués, les systèmes multi-processeurs étroitement couplés, un système d'exploitation pour Ada en temps réel, les limitations de la portabilité des systèmes en Ada et aussi un système pour le traitement des processus périodiques. Une session entière a été consacrée aux tests des applications Ada en temps réel, et une autre aux méthodologies de conception avec présentation de l'environnement visuel Vanna pour la conception des systèmes distribués, basé sur le langage Raddle. Dans la même session, trois chercheurs du SEI ont formulé leurs recommandations pour le génie logiciel en temps réel en Ada et les aspects de la conception orientée-objet dans un environnement en temps réel à multi-processeurs.

L'évaluation de la technologie du compilateur a fait l'objet d'un débat. Pour des considérations générales sur la technologie et l'évaluation d'un compilateur, il faut se référer aux chapitres 4 à 9. Le ACEC (Ada Compiler Evaluation Capability) est un ensemble d'évaluations commandité par le gouvernement des États-Unis qui a été livré pour la première fois à l'automne 1988. Il s'agit d'un ensemble de plus de 1000 tests de performance d'exécution qui est en développement continuel pour couvrir des aspects additionnels d'évaluation. Dans leur état actuel, par rapport à des systèmes pour d'autres langages, les systèmes de compilation Ada sont adéquats pour certains projets mais pas pour tous; leur qualité s'améliore considérablement d'année en année. Une nouvelle génération d'optimisateurs Ada devrait le rendre compétitif pour la majeure partie des applications spécialisées et intégrées à l'équipement, ou pour des machines nues dans un an ou deux. La qualité des générateurs de code pour des systèmes de compilation Ada s'est améliorée constamment à partir des premières implantations qui avaient été livrées au milieu des années 80. Malheureusement, la performance des systèmes d'exécution n'a pas tenu le pas avec les améliorations des générateurs de codes. L'efficacité, la performance et la configurabilité sont des aspects des systèmes d'exécution qui sont examinés par les contracteurs, les développeurs des compilateurs et des groupes comme le ARTEWG. Même si des environnements de développement pour les applications sur les ordinateurs intégrés à l'équipement et de mise au point sont lentement en train de rejoindre les niveaux des systèmes non spécialisés et intégrés aux équipements, de meilleurs outils de développement sont requis. En ce qui concerne les impacts des tests ACVC et ACEC, il faut remarquer que si les

tests ne sont pas conduits avec une attention extrême, presque la moitié des résultats sont étiquetés comme non fiables; ou bien les résultats ne sont pas significatifs, avec toutes les conséquences que cela implique. La version courante de ACEC a des problèmes similaires au ACVC dans le fait qu'il néglige des sujets majeurs (comme le test de capacité) et que les tests qui sont inclus ont des priorités différentes pour les utilisateurs. La plupart des usagers seraient mieux servis en exécutant les évaluations par eux-mêmes et en utilisant leurs applications comme programmes de tests.

Dans la session d'évaluation des systèmes d'exécution, trois communications ont été présentées: une sur la comparaison d'un langage Assembler avec Ada pour des applications intégrées aux équipements en temps réel, une autre sur la sélection d'un compilateur Ada pour un système d'avion intégré à l'équipement en temps réel; la dernière sur une évaluation quantitative des capacités d'Ada de traitement des interruptions.

Dans la session des aspects de structure du langage, des interventions reportaient certains résultats sur l'étude du projet 9X, effectué par le SEI, sur les expériences de l'abstraction de données en Ada et sur la conception-implantation d'une bibliothèque mathématique en Ada. (Le projet 9X a aussi donné lieu d'un débat.) Le but du projet 9X est de reviser le standard ANSI/MIL-STD-1815A du langage de programmation Ada pour les années 1990, afin de réfléter les besoins essentiels courants avec un minimum d'impact négatif et un maximum d'impact positif pour la communauté Ada. En outre, du moment que les usagers Ada ont des bénéfices énormes dans le marché d'Ada à cause du statut multi-standard d'Ada, une coordination internationale est essentielle. La communication avec la communauté Ada a été réalisée avec l'établissement de plusieurs groupes de travail, notamment, du côté du gouvernement, des représentants du DoD, NASA, FAA, NIST et d'autres institutions. D'une façon analogue, l'industrie joue un rôle important dans différents groupes des projets Ada qui incluent le "Distinguished Reviewers Group", le "Requirement Team", le "Mapping/Revision Team" et le "User/Implementer Teams". La coordination internationale est réalisée par l'intermédiaire des groupes de travail associés avec l'organisation ISO et avec les organisations professionnelles Ada-Europe. Regroupant 9 pays et plus de 1000 membres, le projet 9 X est un projet communautaire.

Centre de recherche informatique de Montréal

décembre 1990

The SEI Resident Affiliates Program

April 1988

Priscilla J. Fowler

1. SEI Affiliate Relations

The Affiliate Relations Function establishes and maintains the SEI's corporate and government relationships, with the cooperation and support of SEI's Industry Sector Operations and the SEI Joint Program Office. These relationships take the form of Information Exchange or Information Dissemination affiliation, or Resident Affiliate sponsorship. (There is also a category of affiliation for universities, managed by the Education Program.) Affiliates play a key role in SEI technology transition strategy, and the Affiliate Relations Function facilitates that role in a variety of ways.

Information Exchange Affiliates. Information Exchange Affiliates, who must be part of, or suppliers to, the mission-critical computer resource (MCCR) community, comprise the major constituency sharing information, technology, and needs with the SEI. They do this through response to surveys, participation in workshops on topics such as post deployment software support, Ada, and data rights, and through technical interchange meetings and training sessions. Technology Exchange Agreements allow for confidential exchange of company proprietary technology and collaboration in areas of mutual concern. The *Bridge* is a publication written especially for affiliates, and always includes lists of new SEI Technical Reports and other publications available on request. The annual Affiliates Symposium is another forum for information exchange. It provides an in depth update on SEI projects and activities, as well as a report on the state of software engineering practice and technology.

Information Dissemination Affiliates. Information Dissemination Affiliates are organizations in the business of providing information, continuing education, or professional training to organization in the MCCR community. This is a new category of affiliation used by industry analysts, commercial education and training vendors, and consortia. This form of affiliation is similar to Information Exchange Affiliates in services provided by the SEI. These affiliates, however, differ from Information Exchange Affiliates in that their primary business is technology transfer. They help amplify the SEI's own technology transfer capability.

There is no charge for Information Exchange affiliation, which is a prerequisite to the sponsoring of Resident Affiliates. Nominal charges to defray expenses are made for workshops, training sessions, and the Affiliates Symposium.

Projects. Projects at the SEI are of a nature not typically justifiable in a single sponsor organization, but which may provide significant benefit to a group of sponsors. For example, the Methods and Tools for Real-Time Systems project has drawn extensively on a range of industry and

vendor experience to build a software tool taxonomy and a set of tool selection criteria. The Real-Time Embedded Systems Testbed Project has developed a testbed for use in evaluating new ideas and tools for constructing real-time systems. The testbed is available for use by groups inside and outside the SEI.

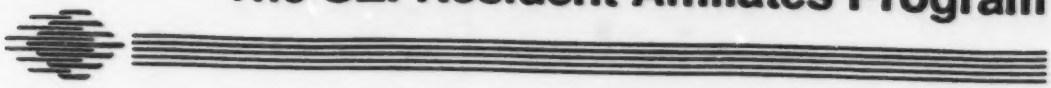
Technology Transition. Resident Affiliates also work to effect transition of their project's findings and those of other SEI efforts to their sponsoring organizations, typically through periodic reports. Data gathering to support technology transition is accomplished formally, through presentations by SEI project members and information exchange at monthly Resident Affiliate meetings and at the weekly SEI Information Seminars, as well as informally, through electronic bulletin boards and personal interaction. In addition, sponsoring organizations are provided the opportunity for special on-site technical interchange meetings with the SEI. Resident Affiliates spend between 10 and 40% of their time on activities specific to technology transition.

Resident Affiliate Sponsorship. Industry and government organizations sponsoring Resident Affiliates supply application domain expertise and software engineering experience, as well as additional resources, to the SEI. Resident Affiliates are senior technical personnel who participate in SEI projects for periods ranging typically from six months to two years.

Determination of Assignments. Resident Affiliate assignments are determined by with the potential sponsor, Affiliate Relations, and the SEI Program Director and Project Manager from the technology area of interest. A memorandum of understanding is prepared jointly by the SEI Director of Affiliate Relations and representatives of the sponsoring organization. The cost of sponsorship is the Resident Affiliate's salary, living allowance, travel, and other usual business expense. SEI provides office space and administrative support, covers project-related expenses such as special training sessions, and provides information on housing, schools, and cost-of-living in Pittsburgh.

Current Resident Affiliates. Currently there are Resident Affiliates from Westinghouse Baltimore Defense Works, GTE, the Naval Air Development Center, IBM, HQ Air Force Space Command, and the Naval Oceans System Center. Projects they are participating in include the Distributed Ada Real-Time Kernel Project, the User Interface Prototyping Project, the Applicable of Reusable Software Components Project, the Ada Embedded Systems Testbed Project, and the Methods and Tools for Real-Time Systems Project.

Other Areas of Emphasis. The Affiliate Relations Function continually works to increase the value and intensity of interaction between the SEI and organizations sponsoring Resident Affiliates. One area of particular emphasis will be the establishing of software engineering process groups (SEPGs) within sponsor organizations. These groups support the Resident Affiliates' technology transition activities by providing focal points in the home organization. They also provide more specific support of pilot technology installation efforts, joint efforts between the SEI and the sponsor which reduce risk by testing the use of a particular technology as well as the process for putting it into place.



The SEI Resident Affiliates Program

Industry and government organizations may sponsor resident affiliates at the SEI to work on technical projects of interest both to the SEI and the sponsoring organization. Resident affiliates supply these projects with application domain expertise and software engineering experience as well as additional resources. They are usually senior technical personnel, who are assigned to the SEI for periods of six months to two years; they may spend half-time to full time on site. Project assignments for resident affiliates are determined by assessing how to make the best use of their particular expertise while meeting the mutual goals and objectives of the SEI and the sponsoring organization.

The SEI conducts projects that would be difficult for a single organization to sponsor but that may provide significant benefits to a group of sponsors. For example, the Methods and Tools for Real-Time Systems project has drawn extensively on a range of industry and vendor experience to build a taxonomy and set of selection criteria for software engineering tools and methods. In addition, SEI projects offer the rare opportunity to collaborate with other industry organizations, university research faculty, and government. The Real-Time Scheduling in Ada Project has sponsored a joint effort of IBM Systems Integration Division, CMU, Verdix, and the Naval Weapons Center that integrates practical experience from a number of different perspectives with scheduling theory and Ada expertise.

Technology Transition. Resident affiliates work to transfer their project's findings and the results of other SEI work to their sponsoring organizations, typically through periodic reports and trips home. Data gathering in support of this effort is accomplished both formally and informally. Monthly resident affiliate meetings provide the opportunity to hear presentations on many SEI projects, and to discuss these with project members. Resident affiliates have access to electronic bulletin boards (available for many projects), SEI Information Seminars, and SEI staff. In addition, resident affiliates are provided both electronic and paper versions of a monthly report summarizing current SEI activities, which they are free to send home. Depending on the emphasis of their assignment, resident affiliates spend 10% to 50% of their time on technology transfer activities.

The SEI has gained a modest working knowledge of software engineering technology transfer, and is now able, as resources permit, to provide resident affiliates with some guidance in planning their technology transfer efforts. The opportunity to attend a new workshop on implementing technological change is also available.

Determination of Assignments. Resident affiliate assignments are established through an iterative process of matching the objectives of the sponsor organization and the SEI. A Memorandum of Understanding is prepared jointly by the SEI Director of Affiliate Relations, the appropriate SEI project manager, and representatives of the sponsoring organization. Costs of sponsorship include the resident affiliate's salary, relocation, living allowance, and travel and other usual business expenses. SEI provides office space and administrative support, covers project-related expenses such as project travel and special training sessions, and provides relocation assistance, including information on housing, schools, and the cost of living in Pittsburgh.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Current Resident Affiliates. Currently there are resident affiliates from Westinghouse Defense and Electronics Center, GTE Government Systems, IBM Systems Integration Division, Computer Sciences Corporation, General Electric, Hughes, the Naval Surface Weapons Center, the Naval Undersea Warfare Engineering Station, CECOM, and the Naval Air Development Center. Projects they are participating in include the Distributed Ada Real-Time Kernel Project, the User Interface Prototyping Project, the Software Reuse Project, the Real-Time Embedded Systems Testbed Project, the Software Architectures Project, the Process Development Project, and the Software Capability Evaluation Project.

For further information, please contact Priscilla Fowler at (412) 268-7748.

May 1989

LISTE DE LA DOCUMENTATION DISPONIBLE SUR LES VENDEURS

La liste suivante représente la documentation disponible au CRIM sur les systèmes Ada commerciaux.

VENDEURS:

Alsys, Burlington, MA

Cadre Technologies Inc, Beaverton, OR

Concurrent Computer Corporation, Tinton Falls, NJ

Digital Equipment Corporation

Encore Computer Corporation, Ft. Lauderdale, FL

Harris Corporation, Fort Lauderdale, FL

Intermetrics, Inc., Cambridge, MA

Meridian Software Systems Inc., Irvine, CA

R.R. Software, Inc.

Rational, Calverton, M.D.

Tartan Laboratories Incorporated, Monroeville, PA

Telesoft, San Diego, CA, DDC-1, Inc., Phoenix, AZ

Texel & Co. Inc., Eatontown, NJ

TLD Systems Ltd., Torrance, California

Verdix Corporation, Chantilly, VA